Дмитрий
Скляров

Искусство защиты и взлома информации

Санкт-Петербург
"БХВ-Петербург"
2004

Содержание

Посвящается

Оксане, Егору и Полине
Саше Каталову
Лене Павловской и Сереже Осокину
Маше Ручке и Игорю Баздыреву
Марине Портновой и Леониду Агранонику
Джеку Палладино (Jack Palladino)
Эдмунду Хинцу (Edmund Hintz)
Полу Хольману (Paul Holman)

И многим другим,
чья поддержка и помощь
помогли выжить и победить.

Введение

Большинство книг по защите информации содержит в себе стройный набор правил, беспрекословное выполнение которых, по идее, должно обеспечить необходимый уровень защиты. Однако, как показывает практика, точное следование правилам далеко не всегда приводит к желаемому результату. Этому есть несколько причин.

Во-первых, все математически строгие доказательства строятся на моделях, которые трудно применимы на практике из-за чрезмерной жесткости накладываемых ограничений. Так, например, легко доказать, что одноразовый блокнот является абсолютно стойким шифром, но при его использовании размер ключа должен быть не меньше размера шифруемых данных и ключ не должен использоваться дважды. На подобные условия согласятся разве что дипломаты, военные или спецслужбы, а для массового использования такой алгоритм не пригоден.

Во-вторых, реальный мир гораздо разнообразней, чем его описывают в книгах, и всегда может возникнуть ситуация, которая не только никому не встречалась на практике, но и даже не приходила в голову. Следовательно, не может быть и полных формальных правил поведения в такой ситуации.

А в-третьих, людям свойственно ошибаться. И программист, реализующий правила безопасности, тоже не застрахован от ошибок. Для обычных программ работоспособность обычно подтверждается путем выполнения серии тестов, проверяющих все основные режимы. Но если программа имеет дело с безопасностью, все обстоит иначе. Адекватное (обеспечивающее должный уровень защиты) поведение программы на всех тестах не гарантирует отсутствия "дыры" в одном из режимов, который не был протестирован. Для получения подобных гарантий требуется выполнить тестирование во всех возможных режимах, что практически нереализуемо.

В этой книге нет универсальных рекомендаций по построению надежных средств защиты, как нет и подробного описания техник, применяемых для

взлома. Внимание читателя направляется на то, какие ошибки чаще всего допускаются в процессе разработки средств защиты, и приводятся примеры реальных систем, которые были взломаны из-за таких ошибок.

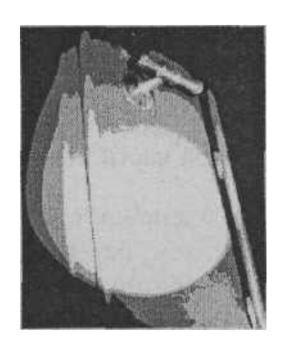
Первая, вводная часть книги призвана дать читателю общее понимание задач информационной безопасности и проблем, возникающих при решении этих задач.

Вторая часть, в основном, посвящена криптографии, т. к. криптография является очень мощным инструментом, без которого построение большинства систем защиты было бы просто невозможно.

В третьей части рассматриваются вопросы защиты программ от несанкционированного тиражирования. Приводятся описания наиболее распространенных приемов защиты, и объясняется, почему эти приемы не всегда работают.

Четвертая часть рассказывает о защите данных. В числе прочих рассматриваются вопросы реализации систем управления цифровыми правами. Дается анализ основных причин, приводящих к появлению ненадежных средств защиты.

В пятой, заключительной части собрана информация о том, как и для чего проводятся исследования программных средств защиты информации. Разработчикам защит полезно знать, какие средства есть в распоряжении противника, чтобы максимально осложнить его работу.

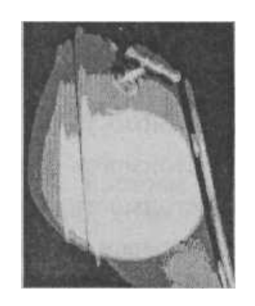


Часть I

КОМУ НУЖНА ЗАЩИТА?

Эта часть вводная. Вряд ли она будет очень интересна профессионалам в области защиты информации, т. к. призвана донести до читателя основные понятия и терминологию предметной области, а также общее представление о том, что может и чего не может информационная безопасность. Однако, наверняка, даже хорошо знакомые с данной темой люди смогут найти для себя в этой части что-то новое.

Глава 1



Общие сведения о защите информации

Перед тем как начинать рассмотрение вопросов защиты информации, стоит более или менее формально определить, что скрывается за терминами "информационная безопасность" и "защита информации". Прежде всего, оба этих словосочетания являются переводом на русский язык английского термина "information security". Словосочетание "информационная безопасность" имеет скорее научный, теоретический окрас, а "защита информации" обычно используется при описании практических мероприятий. Однако, в целом, они являются синонимами, и в книге между ними не будет делаться каких-либо различий.

Мы будем оперировать термином "информация" в максимально широком его понимании. *Информацией* являются любые данные, находящиеся в памяти вычислительной системы, любое сообщение, пересылаемое по сети, и любой файл, хранящийся на каком-либо носителе. Информацией является любой результат работы человеческого разума: идея, технология, программа, различные данные (медицинские, статистические, финансовые), независимо от формы их представления. Все, что не является физическим предметом и может быть использовано человеком, описывается одним словом — информация.

1.1. Что и от чего защищать

До начала рассмотрения различных аспектов защиты информации необходимо выяснить, что и от кого предполагается защищать. Без этого рассуждать о преимуществах и недостатках систем информационной безопасности просто бессмысленно.

1.1.1. Характеристики информации

Прежде всего, у каждой "единицы" защищаемой информации есть несколько параметров, которые необходимо учитывать:

- статичность;
- размер и тип доступа;

- время жизни;
- стоимость создания;
- стоимость потери конфиденциальности;
- стоимость скрытого нарушения целостности;
- стоимость утраты.

Статичность определяет, может ли защищаемая информация изменяться в процессе нормального использования. Так, например, передаваемое по сети зашифрованное сообщение и документ с цифровой подписью изменяться не должны, а данные на зашифрованном диске, находящемся в использовании, изменяются постоянно. Также изменяется содержимое базы данных при добавлении новых или модификации существующих записей.

Размер единицы защищаемой информации может накладывать дополнительные ограничения на используемые средства защиты. Так блочные алгоритмы шифрования в некоторых режимах оперируют порциями данных фиксированной длины, а использование асимметричных криптографических алгоритмов приводит к увеличению размера данных при зашифровании *(см. гл. 5). Тип доступа* (последовательный или произвольный) также накладывает ограничения на средства защиты — использование потокового алгоритма шифрования для больших объемов данных с произвольным доступом требует разбиения данных на блоки и генерации уникального ключа для каждого из них.

Время жизни информации — очень важный параметр, определяющий, насколько долго информация должна оставаться защищенной. Существует информация, время жизни которой составляет минуты, например отданный приказ о начале атаки или отступления во время ведения боевых действий. Содержимое приказа и без расшифровки сообщения станет ясно противнику по косвенным признакам. Время жизни большей части персональной информации (банковской, медицинской и т. п.) соответствует времени жизни владельца — после его смерти разглашение такой информации уже никому не принесет ни вреда, ни выгоды. Для каждого государственного секрета, как правило, тоже определен период, в течение которого информация не должна стать публичной. Однако с некоторых документов грифы не снимаются никогда — это случай, когда время жизни информации не ограничено. Никогда не должна разглашаться информация и о ключах шифрования, вышедших из употребления, т. к. у противника могут иметься в наличии все старые зашифрованные сообщения и, получив ключ, он сможет обеспечить себе доступ к тексту сообщений.

Стоимость создания является численным выражением совокупности ресурсов (финансовых, человеческих, временных), затраченных на создание информации. Фактически, это ее себестоимость.

Стоимость потери конфиденциальности выражает потенциальные убытки, которые понесет владелец информации, если к ней получат неавторизован-

ный доступ сторонние лица. Как правило, стоимость потери конфиденциальности многократно превышает себестоимость информации. По истечении времени жизни информации стоимость потери ее конфиденциальности становится равной нулю.

Стоимость скрытого нарушения целостности выражает убытки, которые могут возникнуть вследствие внесения изменений в информацию, если факт модификации не был обнаружен. Нарушения целостности могут носить различный характер. Они могут быть как случайными, так и преднамеренными. Модификации может подвергаться не только непосредственно текст сообщения или документа, но также дата отправки или имя автора.

Стоимость утраты описывает ущерб от полного или частичного разрушения информации. При обнаружении нарушения целостности и невозможности получить ту же информацию из другого источника информация считается утраченной.

Четыре различных стоимости, перечисленные выше, могут очень по-разному соотноситься друг с другом. Рассмотрим два примера.

Представим следующую ситуацию. Человек при помощи специализированной программы заносит информацию обо всех своих счетах и финансовых операциях в базу данных. Вследствие особенностей налоговой системы подобная практика является обычной для жителей некоторых стран. Стоимость создания подобной базы данных складывается преимущественно из времени, потраченного на ее заполнение актуальными данными. Финансовая информация, по большому счету, является конфиденциальной. Для того чтобы защитить эту информацию от потенциальных похитителей, почти все программы ведения личной финансовой истории, такие как Microsoft Money или Intuit Quicken, позволяют зашифровать базу данных и защитить ее паролем. Утечка информации крайне нежелательна, но однозначно оценить величину возможного ущерба довольно тяжело. Для кого-то потеря конфиденциальности финансовой информации пройдет бесследно, для кого-то создаст значительные трудности. Если в базу будут внесены скрытые изменения, это может привести к ошибкам в налоговой отчетности, а это, в свою очередь, чревато серьезными последствиями, вплоть до уголовного преследования. Ущерб от утраты содержимого зашифрованной базы зачастую оказывается больше ущерба от нарушения конфиденциальности вследствие попадания базы в чужие руки. В таком случае при утере пароля к базе владелец может обратиться в компанию, оказывающую услуги по восстановлению забытых паролей.

В качестве второго примера возьмем смарт-карту, в памяти которой хранится секретный ключ криптосистемы с открытым ключом, используемый как для шифрования, так и для подписи сообщений. Стоимость создания такой карты сравнительно мала. В случае потери конфиденциальности (если противнику удастся извлечь секретный ключ, получить неограниченный доступ

к самой карте или создать ее точную копию) могут наступить весьма тяжелые последствия: противник получает возможность читать все зашифрованные сообщения и подписывать сообщения от имени владельца карты. Скрытое нарушение целостности в данном случае почти не имеет смысла. Даже если противнику удастся подменить в карте секретный ключ, владелец карты не сможет не заметить, что не в состоянии расшифровать старые сообщения, а создаваемая подпись не опознается как принадлежащая ему, т. к. опубликованный открытый ключ не соответствует новому секретному ключу. То же самое произойдет и при утрате карты или ключа. Как видно, в этом случае, несмотря на невозможность повторного создания карты с тем же самым секретным ключом, утрата карты предпочтительнее потери конфиденциальности. Именно поэтому современные смарт-карты не позволяют прочитать секретный ключ стандартными средствами, а при попытке физического вмешательства во "внутренности" карты, данные просто уничтожаются.

1.1.2. Угрозы безопасности

При рассмотрении вопросов безопасности информационных систем практически все авторы выделяют три вида угроз безопасности:

- угрозы конфиденциальности информации;
- угрозы целостности информации;
- угрозы отказа в обслуживании.

Рассмотрим их подробнее.

Нарушение конфиденциальности возникает тогда, когда к какой-либо информации получает доступ лицо, не имеющее на это права. Этот вид угроз, пожалуй, наиболее часто встречается в реальном мире. Именно для уменьшения подобных угроз рекомендуется хранить в сейфах документы, содержащие секретные сведения, а при работе с такими документами вводить специальные защитные процедуры (допуски, журналы регистрации и т. п.).

Нарушение целостности происходит при внесении умышленных или неумышленных изменений в информацию. В реальном мире примером нарушения целостности может являться, например, подделка документов. Чтобы избежать этого, используются специальная бумага (с водяными знаками, голограммами и т. д.), печати и подписи. Для заверки подлинности документов существуют нотариальные службы.

Отказ в обслуживании угрожает не самой информации, а автоматизированной системе, в которой эта информация обрабатывается. При возникновении отказа в обслуживании уполномоченные пользователи системы не могут получить своевременный доступ к необходимой информации, хотя имеют на это полное право.

1.1.3. Потенциальный противник

Теперь, когда мы рассмотрели свойства информации и угрозы ее безопасности, осталось определить, кто может попытаться реализовать эти угрозы.

Очевидно, что отсутствие ключа в замке зажигания и запертая дверь не дадут угнать автомобиль первому попавшемуся прохожему. Но взломщик с набором инструментов и соответствующими навыками легко откроет дверь и заведет двигатель. От такого взломщика может спасти противоугонная система, купленная за несколько сотен долларов. Но и она, с большой вероятностью, окажется бессильной против профессионала, использующего оборудование стоимостью в десятки тысяч долларов и собирающегося угнать не первую попавшуюся, а совершенно определенную машину.

То же самое происходит и при защите информации. Некоторые методы способны обеспечить защиту от рядового пользователя, но оказываются бессильны, если атаку выполняет профессионал. А те средства защиты, которые способны остановить профессионала, не обязательно являются непреодолимым препятствием для правительственного агентства крупной мировой державы.

С правительственными агентствами связан еще один нюанс. Законодательство многих стран содержит статьи, регулирующие использование средств информационной безопасности и, в частности, криптографии. Так в течение многих лет существовали и повсеместно применялись экспортные ограничения правительства США, направленные на запрет продажи за границу программного обеспечения, использующего стойкие криптографические алгоритмы. Разрешенная длина ключа составляла 40 бит, что на момент введения ограничений позволяло защищать информацию от противника-одиночки, т. к. перебор 2^{40} комбинаций на персональном компьютере требовал нескольких десятков, если не сотен лет непрерывной работы процессора. А Агентство Национальной Безопасности США (National Security Agency, NSA) могло, используя имеющийся в наличии парк вычислительных систем, взломать 40-битовый шифр в течение нескольких дней, а то и часов. Суммарная вычислительная мощность, доступная Агентству Национальной Безопасности (АНБ), возможно, самая большая в мире. По некоторым данным, финансирование АНБ превышает суммарное финансирование ЦРУ и ФБР. И при этом про само АНБ известно крайне мало, даже существует полушутливая расшифровка аббревиатуры NSA: No Such Agency (Нет Такого Агентства).

С ростом производительности вычислительных систем 40-битовый ключ стал явно недостаточным, и производителям программного обеспечения пришлось пускаться на различные ухищрения. Так, например, в Lotus Notes для шифрования отсылаемых сообщений использовался 64-битовый ключ,

что обеспечивало высокий уровень стойкости, хотя и не являлось абсолютной защитой. Но интернациональная (экспортная) версия Lotus Notes посылала вместе с каждым сообщением 24 бита из ключа, тем самым уменьшая эффективную длину ключа до 40 бит. Эти 24 бита зашифровывались с использованием открытого ключа, принадлежащего АНБ, и помещались в так называемое *поле сокращения перебора* (Work factor Reduction Field, WRF). Для расшифровки перехваченного сообщения потенциальному противнику необходимо было выполнить перебор 2^{64} возможных ключей, в то время как АНБ могло с помощью известного только ему секретного ключа расшифровать 24 бита, переданные в поле сокращения перебора. После этого АНБ оставалось перебрать только 2^{40} вариантов ключа, что в 16 миллионов раз меньше полного перебора.

При оценке возможностей потенциального противника не стоит упускать из виду и тот факт, что технический прогресс не стоит на месте и каждый день появляются более мощные компьютеры, более эффективные технологии, а иногда и принципиально новые методы атаки. Все это необходимо учитывать при выборе средств защиты для информации с относительно большим сроком жизни.

Если 5 лет назад полный перебор всех возможных комбинаций 40-битового ключа шифрования на одном компьютере считался практически непосильной задачей, то сейчас (в 2004 году) документ в формате Microsoft Word или PDF, защищенный ключом такой длины, может быть расшифрован меньше чем за неделю. В качестве наибольшего достижения, демонстрирующего возможности современных распределенных вычислительных систем, можно назвать отыскание полным перебором 64-битового ключа алгоритма RC5, занявшее 1757 дней (почти 5 лет!) и законченное 14 июля 2002 года.

Еще одной хорошей иллюстрацией прогресса технических средств являются оценки стоимости взлома алгоритма шифрования DES (Data Encryption Standard). DES представляет собой модификацию шифра Lucifer, разработанного компанией IBM и представленного на рассмотрение правительства США в 1975 году. Внесенные в Lucifer изменения, прежде всего, коснулись длины ключа: она была сокращена со 112 до 56 бит по решению АНБ. 23 ноября 1976 года DES был утвержден в качестве федерального стандарта шифрования США и разрешен к использованию во всех несекретных правительственных каналах связи. А 15 января 1977 года было опубликовано официальное описание стандарта, вступившего в силу 6 месяцев спустя.

В статье, опубликованной в 1977 году, известные специалисты в области криптографии Уитфилд Диффи (Whitfield Diffie) и Мартин Хеллман (Martin Hellman) описали проект специализированной вычислительной машины для взлома DES. По их оценкам, она обошлась бы в 20 миллионов долларов

и была бы способна найти нужный ключ максимум за 20 часов работы. В 1981 году Диффи изменил свои оценки, увеличив стоимость до 50 миллионов долларов, а время вскрытия — до двух суток. В 1993 году Майкл Винер (Michael Wiener) спроектировал машину стоимостью 1 миллион долларов, которая должна была находить ключ максимум за 7 часов. Весной 1998 года общественная организация Electronic Frontier Foundation (EFF) продемонстрировала специализированный компьютер стоимостью 250 тысяч долларов, который за 56 часов расшифровал сообщение, зашифрованное DES. В январе 1999 года DES был взломан за 22 часа путем совместного использования 100 тысяч персональных компьютеров и машины, построенной EFF. Сейчас производительность процессоров выросла в несколько раз по сравнению с 1999 годом, и стоимость взлома DES сократилась еще сильнее. Хотя о полном переборе 2^{56} возможных ключей DES на одном персональном компьютере говорить пока не приходится.

1.2. Задачи информационной безопасности

Так с чем же имеет дело информационная безопасность? Далее следует список основных целей и задач, решение которых она должна обеспечить (в скобках приведены английские эквиваленты):

- секретность (privacy, confidentiality, secrecy);
- целостность (data integrity);
- идентификация (identification);
- аутентификация (data origin, authentication);
- уполномочивание (authorization);
- контроль доступа (access control);
- право собственности (ownership);
- сертификация (certification);
- подпись (signature);
- неотказуемость (non-repudiation);
- датирование (time stamping);
- расписка в получении (receipt);
- аннулирование (annul);
- анонимность (anonymity);
- свидетельствование (witnessing);
- подтверждение (confirmation);
- ратификация (validation).

Рассмотрим каждую из перечисленных задач подробнее.

Секретность ~ одна из самых востребованных задач зашиты. Практически у каждого человека или организации найдутся документы, которые ни в коем случае не должны стать всеобщим достоянием, будь то личные медицинские данные, информация о финансовых операциях или государственная тайна. Пока для хранения используются неэлектронные средства (бумага, фотопленка), секретность обеспечивается административными методами (хранение в сейфах, транспортировка в сопровождении охраны и т. д.). Но когда информация обрабатывается на компьютерах и передается по открытым каналам связи, административные методы оказываются бессильны и на помощь приходят методы информационной безопасности. Задача обеспечения секретности, фактически, сводится к тому, чтобы сделать возможным хранение и передачу данных в таком виде, чтобы противник, даже получив доступ к носителю или среде передачи, не смог получить сами защищенные данные.

Целостность — еще одна очень важная задача. В процессе обработки и передачи по каналам связи данные могут быть искажены, как случайно, так и преднамеренно. Также информация может быть изменена прямо на носителе, где она хранится. Проверка целостности просто необходима в ситуациях, когда интерпретация неправильных данных может привести к очень серьезным последствиям, например при возникновении ошибки в сумме банковского перевода или значении скорости самолета, заходящего на посадку. Обеспечение целостности (контроль целостности) заключается в том, чтобы позволить либо утверждать, что данные не были модифицированы при хранении и передаче, либо определить факт искажения данных. То есть никакое изменение данных не должно пройти незамеченным.

Идентификация необходима для того, чтобы отождествить пользователя с некоторым уникальным идентификатором. После этого ответственность за все действия, при выполнении которых предъявлялся данный идентификатор, возлагается на пользователя, за которым этот идентификатор закреплен.

Аутентификация является необходимым дополнением к идентификации и предназначена для подтверждения истинности (аутентичности) пользователя, предъявившего идентификатор. Неанонимный пользователь должен получить возможность работать только после успешной аутентификации.

Уполномочивание сводится к тому, что ни один пользователь не должен получить доступ к системе без успешного выполнения идентификации и последующей аутентификации и ни один пользователь не должен получить доступ к ресурсам, если он не уполномочен на такие действия специальным разрешением.

Контроль доступа — комплексное понятие, обозначающее совокупность методов и средств, предназначенных для ограничения доступа к ресурсам.

Доступ должны иметь только уполномоченные пользователи, и попытки доступа должны протоколироваться.

Право собственности используется для того, чтобы предоставить пользователю законное право на использование некоторого ресурса и, при желании, возможность передачи этого ресурса в собственность другому пользователю. Право собственности обычно является составной частью системы контроля доступа.

Сертификация — процесс подтверждения некоторого факта стороной, которой пользователь доверяет. Чаще всего сертификация используется для удостоверения принадлежности открытого ключа конкретному пользователю или компании, т. к. эффективное использование инфраструктуры открытых ключей возможно лишь при наличии системы сертификации. Организации, занимающиеся выдачей сертификатов, называются удостоверяющими центрами.

Подпись позволяет получателю документа доказать, что данный документ был подписан именно отправителем. При этом подпись не может быть перенесена на другой документ, отправитель не может отказаться от своей подписи, любое изменение документа приведет к нарушению подписи, и любой пользователь, при желании, может самостоятельно убедиться в подлинности подписи.

Неотказуемость — свойство схемы информационного обмена, при котором существует доказательство, которое получатель сообщения способен предъявить третьей стороне, чтобы та смогла независимо проверить, кто является отправителем сообщения. То есть отправитель сообщения не имеет возможности отказаться от авторства, т. к. существуют математические доказательства того, что никто кроме него не способен создать такое сообщение.

Расписка в получении передается от получателя к отправителю и может впоследствии быть использована отправителем для доказательства того, что переданная информация была доставлена получателю не позже определенного момента, указанного в расписке.

Датирование часто применяется совместно с подписью и позволяет зафиксировать момент подписания документа. Это может быть полезно, например, для доказательства первенства, если один документ был подписан несколькими пользователями, каждый из которых утверждает, что именно он является автором документа. Кроме этого датирование широко используется в сертификатах, которые имеют ограниченный срок действия. Если действительный сертификат был использован для подписи, а затем соответствующей службой сертифицирующего центра была проставлена метка времени, то такая подпись должна признаваться правильной и после выхода сертификата из употребления. Если же отметка времени отсутствует, то после

истечения срока действия сертификата подпись не может быть признана корректной.

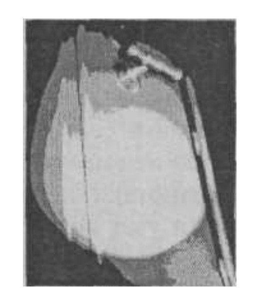
Аннулирование используется для отмены действия сертификатов, полномочий или подписей. Если какой-либо участник информационного обмена или принадлежащие ему ключи и сертификаты оказались скомпрометированы, необходимо предотвратить доступ этого пользователя к ресурсам и отказать в доверии соответствующим сертификатам, т. к. ими могли воспользоваться злоумышленники. Также процедура аннулирования может быть использована в отношении удостоверяющего центра.

Свидетельствование — удостоверение (подтверждение) факта создания или существования информацией некоторой стороной, не являющейся создателем.

Анонимность — довольно редко вспоминаемая задача. Сильным мира сего — правительствам и корпорациям — не выгодно, чтобы пользователь мог остаться анонимным при совершении каких-либо действий в информационном пространстве. Возможно, по этой причине проекты по обеспечению анонимности носят единичный характер и, как правило, долго не живут. Да и средства коммуникации, в подавляющем большинстве, позволяют определить маршрут передачи того или иного сообщения, а значит, вычислить отправителя.

Все перечисленные задачи сформулированы, исходя из потребностей существующего информационного мира. Возможно, со временем часть задач потеряет свою актуальность, по более вероятно, что появятся новые задачи, нуждающиеся в решении.

Глава 2



Основные способы обеспечения защиты

Наверное, потребность в защите информации появилась одновременно с самой информацией. И возможные методы защиты информации почти всегда определялись формой ее представления и предполагаемыми способами использования.

2.1. Общая классификация

В первом приближении все методы защиты информации можно разделить на три класса:

- законодательные;
- административные;
- технические.

Законодательные методы определяют, кто и в какой форме должен иметь доступ к защищаемой информации, и устанавливают ответственность за нарушения установленного порядка. Например, в древнем мире у многих наций были тайные культы, называемые мистериями. К участию в мистериях допускались только посвященные путем особых обрядов лица. Содержание мистерий должно было сохраняться в тайне. А за разглашение секретов мистерий посвященного ждало преследование, вплоть до смерти. Также смертью каралось недозволенное участие в мистериях, даже произошедшее по случайности. В современном мире существуют законы о защите государственной тайны, авторских прав, положения о праве на тайну личной переписки и многие другие. Такие законы описывают, кто и при каких условиях имеет, а кто не имеет право доступа к определенной информации. Однако законодательные методы не способны гарантировать выполнение установленных правил, они лишь декларируют эти правила вместе с мерой ответственности за их нарушение.

Административные методы заключаются в определении процедур доступа к защищаемой информации и строгом их выполнении. Контроль над соблюдением установленного порядка возлагается на специально обученный

персонал. Административные методы применялись многие века и диктовались здравым смыслом. Чтобы случайный человек не прочитал важный документ, такой документ нужно держать в охраняемом помещении. Чтобы передать секретное сообщение, его нужно посылать с курьером, который готов ценой собственной жизни защищать доверенную ему тайну. Чтобы из библиотеки не пропадали в неизвестном направлении книги, необходимо вести учет доступа к библиотечным ресурсам. Современные административные методы защиты информации весьма разнообразны. Например, при работе с документами, содержащими государственную тайну, сначала необходимо оформить допуск к секретным документам. При получении документа и возврате его в хранилище в журнал регистрации заносятся соответствующие записи. Работа с документами разрешается только в специально оборудованном и сертифицированном помещений. На любом этапе известно лицо, несущее ответственность за целостность и секретность охраняемого документа. Схожие процедуры доступа к информации существуют и в различных организациях, где они определяются корпоративной политикой безопасности. Например, элементом политики безопасности может являться контроль вноса и выноса с территории организации носителей информации (бумажных, магнитных, оптических и др.). Административные методы защиты зачастую совмещаются с законодательными и могут устанавливать ответственность за попытки нарушения установленных процедур доступа.

Технические методы защиты, в отличие от законодательных и административных, призваны максимально избавиться от человеческого фактора. Действительно, соблюдение законодательных мер обуславливается только добропорядочностью и страхом перед наказанием. За соблюдением административных мер следят люди, которых можно обмануть, подкупить или запугать. Таким образом, можно избежать точного исполнения установленных правил. А в случае применения технических средств зашиты перед потенциальным противником ставится некоторая техническая (математическая, физическая) задача, которую ему необходимо решить для получения доступа к информации. В то же время легитимному пользователю должен быть доступен более простой путь, позволяющий работать с предоставленной в его распоряжение информацией без решения сложных задач. К техническим методам защиты можно отнести как замок на сундуке, в котором хранятся книги, так и носители информации, самоуничтожающиеся при попытке неправомерного использования. Правда, такие носители гораздо чаще встречаются в приключенческих фильмах, чем в реальности.

Самоуничтожающиеся DVD-диски

Компания Walt Disney объявила о выходе на рынок в августе 2003 года пробной партии самоуничтожающихся DVD-дисков. Такие диски можно будет использовать только в течение 48 часов после вскрытия упаковки. Затем поверхность

диска должна почернеть, что сделает невозможным прохождение лазерного луча DVD-проигрывателя.

Технические методы, пожалуй, имеют наибольшее разнообразие среди всех методов защиты, и эта книга посвящена в основном рассмотрению именно технических методов защиты информации, представленной в цифровой форме.

2.2. Технические методы защиты

Технические методы защиты информации начали разрабатываться очень давно. Так, например, еще в V—IV вв. до н. э. в Греции применялись шифрующие устройства. По описанию древнегреческого историка Плутарха, шифрующее устройство состояло из двух палок одинаковой толщины, называемых сциталами, которые находились у двух абонентов, желающих обмениваться секретными сообщениями. На сциталу по спирали наматывалась без зазоров узкая полоска папируса, и в таком состоянии наносились записи. Потом полоску папируса снимали и отправляли другому абоненту, который наматывал ее на свою сциталу и получал возможность прочесть сообщение. Элементом, обеспечивающим секретность в таком шифрующем устройстве, являлся диаметр сциталы.

Вместе с техническими методами защиты разрабатывались и методы обхода (взлома) зашиты. Так древнегреческий философ Аристотель предложил использовать длинный конус, на который наматывалась лента с зашифрованным сообщением. В каком-то месте начинали просматриваться куски сообщения, что позволяло определить диаметр сциталы и расшифровать все сообщение.

Применительно к информационной безопасности, технические методы защиты призваны обеспечить решение всех задач, перечисленных в *разд. 1.2.* Условно методы решения этих задач можно разделить на те, которые имеют математическое обоснование стойкости к взлому, и те, которые такого обоснования не имеют.

Методы, не имеющие математического обоснования стойкости, проще всего рассматривать как "черный ящик" — некоторое устройство, которому на вход подаются данные, а на выходе снимается результат. Процессы, происходящие внутри "черного ящика", предполагаются неизвестными и неподвластными ни пользователю, ни потенциальному противнику. Собственно, стойкость таких методов основывается именно на предположении, что "ящик" никогда не будет вскрыт и его внутреннее устройство не будет проанализировано. Однако в реальной жизни случается всякое, и иногда или возникает ситуация, при которой раскрывается устройство "черного ящика",

или упорному исследователю удается разгадать алгоритмы, определяющие функционирование зашиты, без вскрытия самого "ящика". При этом стойкость системы защиты становится равна нулю. Методы защиты, функционирующие по принципу "черного ящика", называют Security Through Obscurity (безопасность через неясность, незнание).

Особенность методов защиты, имеющих математическое обоснование стойкости, заключается в том, что их надежность оценивается, исходя из предположения открытости внутренней структуры. То есть предполагается, что потенциальному противнику известны в деталях все алгоритмы и протоколы, использующиеся для обеспечения защиты. И, тем не менее, противник должен быть не в состоянии обойти средства защиты, т. к. для этого ему надо решить некоторую математическую проблему, которая на момент разработки защиты не имела эффективного решения. Однако существует вероятность того, что через некоторое время будет разработан эффективный алгоритм решения математической проблемы, лежащей в основе метода защиты, а это неминуемо приведет к снижению ее стойкости. Большинство методов, имеющих математическое обоснование стойкости, относятся к методам криптографии *(см. гл. 5).* И именно криптографические методы в основном позволяют эффективно решать задачи информационной безопасности.

Теперь рассмотрим, какие методы защиты применяются при решении основных задач информационной безопасности.

2.3. Методы решения задач информационной безопасности

Для обеспечения секретности в подавляющем большинстве случаев используются методы криптографии. Современные алгоритмы шифрования при условии обеспечения секретности ключа предоставляют возможность защиты даже от таких противников, как правительственные агентства. Однако существуют ситуации, когда секретность одних и тех же данных должна обеспечиваться только при попытке выполнения запрещенных действий, а в остальных случаях доступ к данным должен быть свободным.

Защита содержимого DVD-дисков

DVD-диски с кинофильмами могут быть записаны для какого-то определенного региона (Северная Америка, Европа и т. д.) и должны без проблем воспроизводиться на проигрывателях этого региона. Но диск, купленный в США, не должен воспроизводиться на DVD-проигрывателе, купленном в Европе. Также не должно быть возможным извлечение содержимого диска на персональном компьютере. Однако интуитивно понятно, что если данные на DVD-диске зашифрованы и какой-то DVD-проигрыватель может их воспроизвести, значит, диск

вместе с проигрывателем содержат достаточно информации для расшифровки содержимого диска. Отсюда следует вывод, что ключ шифрования присутствует в системе, а значит, выполнив те же действия, что выполняет DVD-проигрыватель, можно расшифровать содержимое диска. Получается, что криптография не способна обеспечить секретность в таких сложных условиях, и защиту приходится основывать на запутывании процесса извлечения содержимого диска. Стоит заметить, что для DVD-дисков задача защиты пока так и не была решена эффективно.

С непреднамеренными нарушениями целостности, возникающими из-за ошибок при передаче данных по каналам связи или сбоев при чтении носителей информации, можно бороться путем добавления избыточности к защищаемой информации, что позволит обнаружить и иногда даже исправить случайно возникающие ошибки. Для этого существует целая теория помехоустойчивого кодирования. В большинстве современных архиваторов (программ для сжатия данных) для контроля целостности распакованного файла используется алгоритм CRC32 (Cyclic Redundant Code, 32-разрядный циклический избыточный код). Перед упаковкой файла для него вычисляется значение CRC32, которое сохраняется вместе со сжатыми данными в архиве. После распаковки снова вычисляется CRC32, и если вычисленное значение не совпало с сохраненным в архиве, файл признается поврежденным. Однако для защиты от умышленного внесения изменений данный метод не подходит — противнику не составит большого труда подкорректировать данные, вычислить от них CRC32 и сохранить исправленные версии в архиве. Даже если противник не имеет возможности исправлять контрольную сумму, использовать CRC32 для защиты не стоит, т. к. этот алгоритм является обратимым — изменением всего четырех байт в файле можно добиться любого нужного значения CRC32. Поэтому для зашиты от нарушений целостности целесообразно использовать стойкие криптографические хэш-функции (hash function), предотвращающие внесение изменений в защищаемые данные, вместе с шифрованием, препятствующим подмене значения хэш-функции. Для этих целей можно использовать шифрование как с секретным, так и с открытым ключом.

При идентификации, как правило, не возникает особенных сложностей: пользователь предъявляет свой идентификатор, а система принимает его. Идентификатором может являться вводимое с клавиатуры имя пользователя, информация, содержащаяся на магнитной полосе пластиковой карты или в памяти смарт-карты, или какой-либо биометрический показатель (отпечаток пальца, форма руки, голос, рисунок радужной оболочки глаза и т. д.). Но почти всегда сразу за идентификацией следует аутентификация, т. к. корректность идентификатора, за исключением биометрического, не гарантирует, что он предъявлен владельцем, а не неким посторонним лицом.

2 Зак. 1310

Аутентификация заключается в том, что пользователь, предъявивший идентификатор, вводит некоторую, известную только ему, секретную информацию, которая после некоторого преобразования передается проверяющей стороне. Это может быть пароль, PIN-код (Personal Identification Number, персональный идентификационный номер) и т. п. Проверяющая сторона на основе имеющейся у нее информации (хэша пароля, зашифрованного значения PIN-кода) принимает решение об аутентичности (подлинности) пользователя.

Правильное решение задач идентификации и последующей аутентификации включает в себя решение нескольких подзадач.

- Обеспечить невозможность успешного повторения идентификации и аутентификации путем перехвата сетевого трафика и повторной отсылки правильных ответов. Для этого используется схема запрос/ответ (challenge/response), когда проверяющая сторона присылает некоторый случайный запрос (challenge), который используется при аутентификации для преобразования введенных пользователем данных перед отправкой их проверяющей стороне. При таком подходе перехват сетевого трафика оказывается бесполезным — проверяющая сторона каждый раз присылает новый запрос, на который существует единственный правильный ответ, который невозможно быстро вычислить, не зная секретной информации (пароля или PIN-кода).

Аутентификация сетевых подключений в Windows 95/98

5 января 1999 года компания LOpht опубликовала статью о найденной уязвимости в реализации схемы запрос/ответ при подключении сетевых ресурсов Windows 95/98. Как оказалось, при попытках подключения Windows 95/98 посылает один и тот же запрос в течение примерно 15 минут, и за это время противник может подключиться к сетевому ресурсу от имени того пользователя, чью попытку аутентификации удалось перехватить.

- Обеспечить невозможность эффективного получения секретной информации, вводимой пользователем при аутентификации, в случае взлома проверяющей стороны. Для этого проверяющая сторона должна хранить не копию секретной информации, вводимой пользователем, а результат применения к ней стойкой криптографической хэш-функции.

Хэши паролей для LANMAN-аутентификации

В таких операционных системах, как Windows NT/2000, могут храниться две версии хэшей пароля. Одна версия используется собственными средствами безопасности Windows NT, а другая нужна для обеспечения совместимости с протоколом аутентификации LANMAN, применяемым, в частности, в Windows 95/98.

Собственный хэш Windows NT достаточно стойкий, и по значению хэша практически невозможно найти пароль, использующий цифры, буквы в разных регистрах и знаки препинания и имеющий длину более 8 символов. Однако процедура вычисления хэша LANMAN имеет несколько особенностей, которые значительно ослабляют уровень защиты.

При вычислении хэша LANMAN-пароль, длина которого не должна превышать 14 символов, разбивается на 2 части по 7 символов, и хэш для каждой части вычисляется отдельно. Таким образом, при подборе пароля максимальная длина проверяемых слов составляет всего 7 символов, что делает возможным даже полный перебор всех вариантов пароля.

Если пароль не длиннее 7 символов, то вторая часть остается пустой и порождает всегда одно и то же значение хэша. Это позволяет по второй половине хэша сразу определить пароли, которые короче 8 символов.

Так как обе части пароля обрабатываются независимо, для паролей длиной от 8 до 12 символов вторая часть может быть найдена максимум перебором всех пятисимвольных комбинаций, что не займет очень много времени. Иногда знание окончания пароля позволяет угадать все слово.

Перед вычислением хэша все буквы пароля переводятся в заглавные, что примерно в 9,4 раза сокращает количество возможных комбинаций (при использовании всех символов ASCII).

Функция вычисления хэша LANMAN не использует подмешивание "соли" (salt) — случайной несекретной величины, делающей значение хэша уникальным для каждого пользователя, даже если пароли совпадают. Эта особенность позволяет тратить на подбор пароля для любого числа пользователей практически одинаковое время, т. к. на каждого дополнительного пользователя добавляется лишь одна операция сравнения, которая выполняется в тысячи раз быстрее, чем вычисление хэша.

После того как по хэшу LANMAN подобран пароль, можно за короткое время подобрать и пароль NT, если его длина не превышает 14 символов. На это потребуется не более 2^{14} вариаций прописных и строчных букв, т. е. не более 16 384 комбинаций.

- Минимизировать вероятность ошибок первого рода (отказ в доступе легитимному пользователю) и второго рода (предоставление доступа пользователю, не имеющему на это права). При использовании биометрики для идентификации и/или аутентификации ошибки первого и второго рода играют значительную роль, т. к. биометрика при оценке степени совпадения измеренной биофизической характеристики и ранее сохра-

ненного ее значения опирается на статистические методы. Если при настройке верификатора ослабить требования к точности совпадений, нелегальному пользователю будет легче случайно проникнуть в систему. Если же требования к точности совпадения повысить, легальные пользователи достаточно часто будут получать отказ в доступе. Хорошо настроенная система идентификации по отпечаткам пальцев необоснованно отказывает в доступе примерно в одном случае из 50 и ошибочно пропускает одного нелегального пользователя из миллиарда.

Плюсы и минусы биометрических систем идентификации

В качестве достоинств биометрических систем называют, прежде всего, то, что только биометрика аутентифицирует именно человека, а не пароль или устройство вроде смарт-карты. При этом биометрический идентификатор ничего не стоит пользователю, и его нельзя забыть, потерять или передать другому лицу.

Правда, сами биометрические системы пока довольно дороги, и их точность может зависеть от психофизического состояния человека (влажность кожи, охрипший голос, неустойчивый почерк из-за травмы руки и т. д.). Но существует еще один аспект использования биометрики, который является обратной стороной невозможности потерять биометрический идентификатор и о котором часто забывают. Если учетная запись некоторого пользователя оказалась скомпрометирована (например к ней подобрали пароль), сменить пароль или создать новую учетную запись не составит большого труда. Если же по каким-то причинам оказался скомпрометирован биометрический идентификатор (например кто-то научился подделывать голос или отпечатки пальцев), у владельца этого идентификатора нет никакой возможности его сменить.

Биометрика может применяться либо для идентификации совмещенной с аутентификацией, либо только для аутентификации. При первом подходе снятая биометрическими методами характеристика сравнивается с учетными записями всех пользователей, зарегистрированных в системе, и если точность совпадения превышает установленное значение, пользователь считается идентифицированным и аутентифицированным. Второй подход требует предоставления идентификатора (имени пользователя, пластиковой карты), а биометрические методы лишь подтверждают его аутентичность. Этот способ хоть и является менее удобным для пользователей, имеет несколько преимуществ. Во-первых, требуется сравнить снятую характеристику только с одним сохраненным значением, а не со всеми. Это может оказаться очень существенным, когда число зарегистрированных пользователей измеряется тысячами. А во-вторых, при наличии традиционного идентификатора гораздо проще пресекать многократные попытки входа в систему нелегального пользователя — достаточно ввести ограничение на максимальное количество попыток входа одного пользователя

за фиксированный период времени (например 3 раза в течение 5 минут). Если же явно выраженный идентификатор отсутствует, можно ограничивать только интенсивность попыток идентификации на одном рабочем месте, но в некоторых случаях (например на проходной большого завода) это нецелесообразно.

При решении таких задач информационной безопасности, как уполномочивание, контроль доступа и обеспечение права собственности, не требуется что-либо защищать. Права доступа к различным объектам определяются формальными правилами в соответствии с используемой моделью управления доступом.

Существует две основных модели разграничения доступа: мандатная и дискреционная.

Мандатное разграничение доступа заключается в том, что компьютерные ресурсы разделяются на группы в соответствии с уровнями их конфиденциальности. Каждому ресурсу присваивается классификационная метка, задающая назначенный ему уровень. Полномочия пользователя задаются максимальным уровнем конфиденциальности информации, доступ к которой ему разрешен. В соответствии с этим разрешается доступ только к данным, уровень секретности которых не выше уровня полномочий пользователя.

Дискреционное управление доступом — это метод ограничения доступа к компьютерным ресурсам, основанный на учете прав пользователя или группы, в которую этот пользователь входит. Использование данной модели позволяет для каждого пользователя или для каждой группы пользователей явно задать полномочия, указав список ресурсов (логические диски, элементы файловой структуры, порты ввода-вывода и т. д.), к которым разрешен доступ, а также права доступа к этим ресурсам. Один из видов полномочий дает пользователю возможность передать права доступа любому другому пользователю или группе.

Существуют и иные виды разграничения доступа, не попадающие ни под мандатную, ни под дискреционную модель. Но подробное рассмотрение систем разграничения доступа выходит за рамки книги.

Задачи сертификации в настоящее время встречаются повсеместно, и для решения этих задач используется криптография с открытым ключом. При выдаче сертификата удостоверяющая сторона гарантирует, что содержимое сертификата является подлинным и может быть использовано при выполнении условия правильности сертификата. Чаще всего сертификаты выдаются на открытые ключи и исполняемые файлы.

При использовании криптографии с открытым ключом большой проблемой является доверие к подлинности открытого ключа адресата. Сам открытый

ключ может быть создан любым человеком, и если он получен не из стопроцентно безопасного источника (например лично из рук адресата), нельзя быть полностью уверенным в том, что это не подделка, выполненная противником. Использование сертификатов открытых ключей позволяет решить эту проблему. Достаточно иметь один или несколько так называемых *корневых сертификатов* — сертификатов открытых ключей, принадлежащих базовым удостоверяющим центрам и подлинность которых может быть проверена множеством способов. Корневой сертификат может быть использован для удостоверения подлинности другого открытого ключа, доверие к которому будет основываться на доверии к базовому удостоверяющему центру, а не к источнику получения ключа. Сертификация также может быть выполнена цепочкой удостоверяющих центров, что не снижает доверия к сертифицируемому открытому ключу, если, конечно, ни один из центров сертификации, участвовавших в цепочке, не был скомпрометирован.

Сертификация исполняемых файлов позволяет подтвердить, что файл был создан именно разработчиком, которому принадлежит сертифицирующий ключ. Это является своеобразной гарантией безопасности: если при использовании сертифицированного исполняемого файла возникают проблемы, всегда можно найти, к кому обращаться с вопросами. А в некоторых случаях разрешено использование исключительно сертифицированных модулей. Так, например, любой криптопровайдер (Cryptographic Service Provider, CSP — модуль, отвечающий за поддержку криптографических функций через стандартный программный интерфейс), используемый в современных версиях Windows, должен быть предварительно сертифицирован компанией Microsoft.

*Ключи **RSA** в библиотеке ADVAPI32.DLL*

В январе 2003 года в конференции **fido7.ru.crypt** появилось сообщение, касающееся процедуры подписания криптопровайдеров в Windows. В сообщении утверждалось, что отсутствие желания каждый раз обращаться в Microsoft и тратить несколько дней на подписание разрабатываемого модуля подтолкнуло к поиску более эффективного решения. Это решение заключалось в факторизации (разложении на простые сомножители) модуля 512-битового ключа RSA, используемого при проверке подписи и хранящегося в библиотеке ADVAPI32.DLL. По утверждению автора сообщения для факторизации потребовалось чуть больше двух месяцев. Вычисления выполнялись на кластере из 10 машин с процессорами Pentium-III, работающими на частотах от 500 до 1200 МГц. Однако строгих доказательств выполненной факторизации, похоже, опубликовано не было.

На самом деле, в библиотеке ADVAPI32.DLL операционных систем семейства NT (Windows NT, 2000 и XP) находится не один, а целых три RSA-ключа, замас-

кированных одинаковым образом. Два из них имеют длину 1024 бита, а один действительно 512 бит.

Кстати, в 1999 году один из разработчиков компании Cryptonym, анализируя отладочную информацию из Service Pack 5 для Microsoft Windows NT 4, обнаружил символьные метки для двух ключей. Одна из меток называлась "KEY", а другая — "NSAKEY", что недвусмысленно дает понять, кто является владельцем второго ключа.

Подпись, как и сертификация, реализуется средствами криптографии с открытым ключом. Обычно подписанию подвергаются документы, но подписывать можно что угодно, в том числе и исполняемые файлы. Отличие от сертификации здесь в том, что при подписи подтверждение подлинности берет на себя сам автор, а при сертификации гарантии и ответственность ложатся на плечи более высокой инстанции.

Подписание модулей расширения для программ семейства Adobe Acrobat

Программы семейства Adobe Acrobat имеют возможность подключать модули расширения (plug-ins), предназначенные для увеличения функциональности. Чтобы модуль расширения мог быть загружен в программу Adobe Acrobat Reader, он должен быть подписан разработчиком. А в некоторых режимах, сопряженных с поддержкой DRM (Digital Rights Management, управление цифровыми правами), разрешается загрузка только модулей, сертифицированных и подписанных компанией Adobe.

Однако исследователями компании ElcomSoft было установлено, что при проверке сертификата модуля расширения участвуют только некоторые поля заголовка переносимого исполняемого файла, что дает возможность вносить в файл изменения, не нарушающие целостность сертификата. Это приводит к возможности коррекции исполняемого кода таким образом, чтобы модуль расширения начал выполнять любые действия, включая опасные.

Описание данной уязвимости было опубликовано CERT (Computer Emergency Response Team, бригада неотложной компьютерной помощи), и Adobe обещала устранить дефект в еще не вышедшей на тот момент версии Acrobat Reader 6. Но Acrobat Reader 6 уже вышел, и он продолжает загружать модули расширения, подписанные разработчиками старым уязвимым способом.

Решение таких задач, как неотказуемость, расписка в получении, свидетельствование и датирование, тесно связано с решением задач сертификации и

подписи и тоже обеспечивается методами асимметричной криптографии.

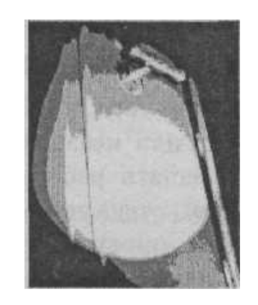
Аннулирование заключается в обновлении списков отозванных сертификатов (Certificate Revocation List, CRL) и списков отмены удостоверяющих центров (Authority Revocation List, ARL). Самым сложным здесь является необходимость обеспечить регулярную и своевременную доставку списков отмены до того, как скомпрометированный ключ будет применен со злым умыслом.

Компрометация сертификатов Microsoft Corporation

30 и 31 января 2001 года удостоверяющий центр компании VeriSign выдал два цифровых сертификата лицу, выдавшему себя путем обмана за работника компании Microsoft. Эти сертификаты могут быть использованы для подписания компонентов ActiveX, макросов Microsoft Office и других исполняемых модулей. VeriSign добавил эти сертификаты в свой список отозванных сертификатов сразу же после обнаружения обмана. Но сертификаты, выпускаемые VeriSign для подписания исполняемого кода, не содержат указания на центр распространения списков отмены (CRL Distribution Point, CDP). Из-за этого программное обеспечение Windows не способно автоматически получить информацию о том, что сертификат был отозван, пока Microsoft выпустит, а пользователь не установит соответствующее исправление.

Для обеспечения анонимности разработаны специальные криптографические протоколы. Они позволяют выполнять такие операции, как тайное компьютеризированное голосование и анонимные не отслеживаемые деньги для оплаты товаров и услуг через Интернет.

Глава 3



Когда защиты слишком много

Один из важных вопросов, нуждающихся в рассмотрении, заключается в том, стоит ли усиливать защиту, если для этого есть хоть какая-то возможность, или может возникнуть ситуация, в которой усиление защиты пойдет не на пользу, а во вред?

До ответа на этот вопрос стоит отметить, что существуют две принципиально разных категории продуктов, использующих средства защиты информации. К первой категории относятся такие продукты, для которых обеспечение информационной безопасности — первоочередная задача. Во вторую категорию попадают прикладные продукты из любых других, явно не связанных с обеспечением безопасности областей, но по тем или иным причинам нуждающиеся в средствах защиты.

При разработке решений, относящихся к первой категории, все должно быть нацелено именно на обеспечение максимальной степени защиты, пусть даже в ущерб другим характеристикам. Такие параметры, как удобство использования и быстродействие, являются несущественными, если задача обеспечения безопасности не решена полностью и простора для компромиссов здесь нет. Ненадежное с точки зрения безопасности решение не должно использоваться, даже если оно в 10 раз удобнее и в 100 раз быстрее, чем любая доступная альтернатива.

С продуктами, относящимися ко второй категории, все обстоит иначе. Для них обеспечение безопасности не является основной задачей. Например, Microsoft Word поддерживает возможность шифрования документов, но при этом Word — текстовый редактор, а не средство для защиты файлов с помощью криптографии. Поэтому к нему предъявляются иные требования, чем к системам обеспечения безопасности.

Прежде всего, обеспечение защиты не должно создавать значительных неудобств. Если на дверь квартиры установить 12 замков, злоумышленнику, наверное, будет труднее проникнуть внутрь. Однако человеку, живущему в этой квартире, придется слишком много времени тратить на отпирание

и запирание всех замков, что с большой вероятностью приведет к недовольству существующим положением вещей.

Кроме того, в процессе эксплуатации изделия могут случаться непредвиденные или незапланированные сбои. Любой замок в какой-то момент может перестать работать. Причины могут быть совершенно разные: заводской брак, отсутствие смазки или ржавчина, сточившийся или погнутый ключ и даже просто проходящий мимо школьник, ради развлечения засунувший спичку в механизм замка. Но результат будет один — замок не удастся отпереть, и человек не сможет попасть в свою квартиру, хотя имеет на это полное право и даже не терял ключи. Интуитивно понятно, что чем больше замков (или уровней защиты) используется, тем больше вероятность отказа хотя бы одного из них. Но отказа любого элемента системы достаточно, чтобы вся система перестала функционировать.

Еще один аспект — эффект "слабого звена". Общая степень защищенности системы определяется уровнем безопасности, который обеспечивает самый слабый узел защиты. Можно устанавливать сколько угодно дополнительных замков в деревянную дверь, но степень защиты все равно будет определяться прочностью двери — самого слабого элемента.

Замки всего лишь пример из реального мира. Но пространство информационных технологий подчиняется тем же самым принципам, и найти схожие примеры совсем несложно.

3.1. Неудобства для пользователя

Один из распространенных способов защиты программ от несанкционированного копирования (тиражирования) связан с использованием регистрационного кода, который пользователь должен ввести в окне регистрации для получения работоспособной версии программы. Регистрационный код, как правило, вычисляется автором (владельцем прав) или распространителем (продавцом) программного продукта на основании предоставленной пользователем информации (например его имени и названия компании, в которой он работает). Процедура вычисления может быть основана на некотором секретном алгоритме, разработанном автором, или на криптографии с открытым ключом. В обоих случаях в программе должен присутствовать обратный алгоритм, позволяющий проверить правильность регистрационного кода. Но в случае применения криптографии с открытым ключом, зная алгоритм проверки, математически сложно полностью воссоздать алгоритм вычисления регистрационного кода. А при использовании секретного алгоритма чаще всего обращение алгоритма проверки после его извлечения из программы не является математически сложной задачей.

Для обеспечения заданного уровня стойкости необходимо использовать ключи не короче определенной длины. Но использование длинных ключей может создавать неудобства пользователю.

Длинные регистрационные ключи

В случае применения криптографии с открытым ключом существует минимальный размер блока, который получается в результате вычисления значения криптографической функции. Для алгоритма RSA-1024 (с ключом длиной 1024 бита) размер зашифрованного блока составляет также 1024 бита или 128 байт. Так как с клавиатуры, в общем случае, можно ввести не всевозможные символы, то двоичные данные, полученные в результате шифрования, преобразуют в текст, например с помощью алгоритма MIME64. При этом каждые 6 входных бит преобразуются в 8, т. е. размер блока данных увеличивается на одну треть и превращается из 128 в 171 символ. Даже если максимально использовать все 95 символов ASCII, вводимых со стандартной клавиатуры, произвольное двоичное сообщение длиной 128 байт удастся закодировать только в 156 символов.

Таким образом, пользователь, не имеющий доступа к Интернету, может оказаться перед необходимостью ввести с клавиатуры присланную ему по факсу или бумажной почтой строку длиной более 150 символов. Более того, строка будет состоять не из простых для восприятия слов, а из бессмысленной смеси всевозможных печатных знаков, в которых очень легко ошибиться. А если пользователь, находящийся вдали от факсимильного аппарата, захочет, чтобы ему продиктовали его регистрационный код по телефону? Вряд ли человек, прошедший через подобную процедуру, порекомендует кому-нибудь из своих друзей приобрести лицензию на такую программу.

3.2. Снижение производительности

Производительность (скорость выполнения) тоже довольно часто страдает, если защищаемая программа злоупотребляет функциями зашиты.

Чрезмерное потребление процессорного времени

В одной из программ, играющих в Gomoku (по-другому TicTacToe — крестики-нолики на большом поле), для защиты от исследования применялся следующий подход. Программа была упакована и зашифрована, и при загрузке в память сама себя расшифровывала и подготавливала к работе. При этом вместо прямых вызовов функций из библиотек Windows, таких как kernel32.dll, user32.dll или gdi32.dll, выполнялись вызовы небольших процедур. Каждая из

этих процедур вычисляла контрольную сумму определенного фрагмента программы в памяти и на основании полученного значения вычисляла адрес библиотечной функции, которой необходимо передать управление. Если в программе происходили малейшие изменения, например в отладчике задавалась точка останова путем изменения одного байта, контрольная сумма оказывалась иной и переход осуществлялся по неверному адресу.

Однократное выполнение подсчета контрольной суммы не занимает слишком много времени. Однако при отображении элементов интерфейса обычно вызывается много библиотечных функций подряд. Все это приводило к тому, что программа выводила данные на экран с видимыми невооруженным глазом задержками.

Стоит отметить, что алгоритм просчета оптимального хода в игре не требует обращения к Win32 API, а значит, применение такой защиты не сказывается на времени, необходимом компьютеру для выбора лучшего варианта. В противном случае программой с подобной защитой было бы невозможно пользоваться.

Также задержки в процессе выполнения часто свойственны системам, использующим аппаратные ключи для защиты от несанкционированного тиражирования. Дело в том, что разработчики ключей защиты любят вводить задержку в аппаратную часть ключа для того, чтобы снизить скорость и эффективности атаки методом перебора. Также разработчики рекомендуют выполнять случайные, фиктивные обращения к ключу с целью затруднить построение компактной таблицы для эмуляции ответов ключа. Если программа при проверке корректности защиты делает большое число последовательных обращений к ключу, суммарное время задержки оказывается весьма ощутимым, и в некоторых "особо выдающихся" программах пользователя заставляют ждать несколько секунд, пока не завершатся все запросы к ключу. Если же используется ключ, установленный не на локальном компьютере, а где-то в сети, ко времени ожидания ответа от ключа добавляется время, затрачиваемое на поиск сервера, обслуживающего ключ, и обмен с ним запросами и ответами. К счастью, большое количество обращений к ключу обычно выполняется только при запуске программы, а в процессе работы производятся только отдельные обращения, но даже единичное длительное ожидание при запуске программы явно не доставляет пользователю особой радости.

3.3. Сбои в системе защиты

Увеличение сложности защитного механизма, возможно, способно повысить уровень защищенности и снизить вероятность несанкционированного дос-

тупа до пренебрежимо малой величины. Но при этом резко возрастет вероятность отказа в доступе легитимному пользователю.

В Интернете в форумах, посвященных защите информации, иногда появляются люди, считающие, что им удалось придумать новый, очень хороший способ защиты от несанкционированного распространения и использования программных продуктов. Грубо метод описывается примерно следующими действиями:

- при установке программы необходимо собрать как можно больше информации о среде функционирования (модель и частота процессора, имя производителя материнской платы, серийный номер жесткого диска и т. д.);
- в различные области файловой системы (INI-файлы, каталог Windows, случайно выбранные каталоги на диске) и системные области операционной системы (реестр Windows) заносятся записи-метки, позволяющие контролировать неизменность среды функционирования и целостность других меток;
- при каждом запуске программы встроенная система защиты произвольным образом выбирает алгоритм проверки корректности одной или нескольких меток, т. е. проверяется целостность метки и соответствие текущих параметров среды функционирования сохраненным в метке значениям;
- если отклонение от идеала (состояния, в котором была система сразу после установки) превышает некоторый допустимый предел, то включается ответная функция защиты. Причем совершено не обязательно пользователю будет сообщено о нарушении зашиты — программа может казаться вполне работоспособной, но некоторые операции будут выполняться неадекватно, например па экране будет выводиться не все, что должно, а при печати наоборот появятся лишние объекты.

Все эти меры принимаются для того, чтобы максимально усложнить потенциальному взломщику жизнь. Особой изюминкой считается последняя часть. Ведь вполне логично, что взломанная программа не обязана работать правильно.

По большому счету, на схожих принципах работает почти любая защита от несанкционированного тиражирования, использующая привязку к компьютеру. Вся разница только в количестве примененных приемов, затрудняющих деактивацию защитных механизмов.

Ложное срабатывание защиты

Современные вычислительные системы стали настолько сложны и разнообразны, что гарантировать безошибочность выполнения в них даже самых про-

стых программ невозможно. Основываясь на этом утверждении можно предположить, что при выполнении любой программы рано или поздно произойдет какой-нибудь сбой. И чем сложнее программа (или ее защита), тем больше вероятность возникновения ошибки.

Кроме того, на компьютере, где функционирует защищенная программа, может быть установлено множество других программ, и их поведение в общем случае почти безопасно, но совершенно непредсказуемо — только авторы этих программ могут догадываться, что и как должно происходить при выполнении их кода. А компьютер, зараженный вирусом, может совершать и деструктивные действия.

И наконец, за компьютером сидит пользователь — человек, который имеет право ошибаться, может сознательно или случайно что-нибудь изменить в окружении защищенной программы и вообще не обязан быть экспертом в вычислительной технике.

Все приведенные выше особенности среды функционирования защищенной программы можно свести к одному утверждению: даже корректно установленная и лицензированная программа по многим причинам может работать не так, как предполагают разработчики. То же самое относится и к защитным механизмам, хотя есть и небольшое отличие. Если не работает какая-то функция в самой программе, все остальные функции остаются доступными. Если же происходит ложное срабатывание механизмов защиты и легальному пользователю отказывают в доступе, работать с программой становится невозможно в принципе.

Когда пользователь замечает, что программа ведет себя не так, как ожидается, он обращается в службу технической поддержки. И в задачу этой службы, как правило, входит определить, действительно ли при выполнении происходит ошибка или пользователь просто чего-то не понял. Если же факт возникновения ошибки подтвержден, необходимо найти и устранить порождающие ее причины.

И тут служба технической поддержки оказывается перед сложной задачей — как по полученной от пользователя (и почти всегда неполной) информации определить, что же случилось на самом деле? Если нарушение защиты проявляется как неадекватное поведение программы, то с чем столкнулся пользователь: с реакцией защиты или действительно с ошибкой в программе? Даже если пользователь получает недвусмысленное сообщение о нарушении защиты, но сама защита очень разветвленная и многоуровневая, как выявить истинную причину сбоя, если у других людей все работает, а у него — нет?

3.4. Материальные затраты пользователей

Когда человек приобретает программный продукт, защищенный от несанкционированного тиражирования, он хочет купить только сам продукт, а защиту ему навязывают. Более того, он за эту защиту еще и вынужден платить — аппаратный ключ, идущий в комплекте с программой, стоит даже в небольших партиях несколько десятков долларов. Разумеется, разработчик в каждую продаваемую копию закладывает стоимость ключа. Если защита не использует аппаратных элементов, все равно на ее разработку или приобретение были затрачены материальные ресурсы, и эти затраты необходимо окупить. Следовательно, стоимость продукта для конечного пользователя все равно повышается.

Закон Российской Федерации "О защите прав потребителей"

Второй пункт 16 статьи Закона РФ "О защите прав потребителей" гласит:

"Запрещается обусловливать приобретение одних товаров (работ, услуг) обязательным приобретением иных товаров (работ, услуг). Убытки, причиненные потребителю вследствие нарушения его права на свободный выбор товаров (работ, услуг), возмещаются продавцом (исполнителем) в полном объеме".

Не являясь юристом, трудно дать точную правовую оценку ситуации, но с точки зрения здравого смысла аппаратный ключ защиты является не чем иным, как навязанным дополнительным товаром. Утверждение, что ключ — неотъемлемая часть программного продукта, кажется несостоятельным, т. к. легко доказать, что разработчиками были затрачены специальные усилия для создания зависимости работоспособности программы от наличия ключа, а сам ключ никак не улучшает потребительские качества продукта.

Несмотря на то, что данная статья присутствует в Законе давно и программы, защищенные аппаратными ключами, продаются не один год, о прецедентах пока не известно.

3.5. Здравый смысл

Если, приобретая защищенную программу, пользователь оказывается перед необходимостью тратить усилия на создание условий, в которых подсистема защиты согласится работать, ценность программы для него резко снижается. Ведь деньги платились ради автоматизации некоторого процесса, а не для "развлечений" в виде потерянного времени.

Вполне разумным кажется следующий подход: лучше ослабить защиту и допустить возможность существования нелегальных пользователей, чем создавать неудобства честным потребителям из-за слишком строгой защиты. Один легальный пользователь, обиженный неудачно реализованной защитой, способен создать продукту изрядную антирекламу.

Разумеется, существуют исключения. Например, если программный продукт не имеет реальных конкурентов (в силу своей уникальности или сильно заниженной по сравнению с другими цены), пользователь, которому этот продукт нужен для работы, все равно согласится на любые дискриминирующие условия.

Однако в большинстве случаев разработчикам рано или поздно приходится считаться с мнением пользователей, чтобы их не потерять.

Активация программных продуктов

Компания Intuit Inc., специализирующаяся на разработке программного обеспечения для учета финансов, в числе прочих выпускает и TurboTax — программу для подготовки налоговой отчетности. В версии TurboTax для 2002 налогового года Intuit, оправдывая свои действия борьбой с использованием нелицензионных версий TurboTax, реализовала систему активации, подобную той, что используется сейчас во многих продуктах корпорации Microsoft. По грубым оценкам для 98 % пользователей активация через Интернет не создала практически никакого неудобства. Но у 2 % возникли сложности. И к тому же активация позволяла полноценно использовать TurboTax только на одном компьютере. В результате многие пользователи отказались от использования TurboTax в пользу его давнего конкурента — программного продукта TaxCut, выпускаемого компанией H&R Block Inc. А как минимум один из пользователей подал судебный иск против компании Intuit в защиту прав всех пользователей TurboTax для 2002 налогового года. Иск основывался на утверждении, что Intuit занимается нечестным бизнесом, недостаточно полно информируя о механизмах активации и последствиях их использования клиентов до того, как они приобретут программный продукт.

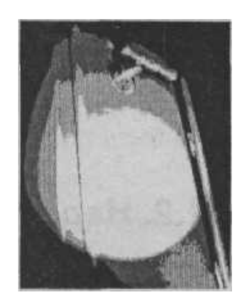
Негативная реакция пользователей заставила Intuit заявить в мае 2003 года об исключении процедуры активации из будущих версий TurboTax. Кстати, компания H&R Block, которая сначала сама планировала ввести процедуру активации для своих продуктов, после неудачи Intuit отказалась от этих планов. Более того, факт отсутствия активации в TaxCut использовался в слогане рекламной кампании как очевидное преимущество.

Корпорация Microsoft нашла компромиссное решение для активации. Основной массе пользователей при установке, например, любой современной версии операционной системы Windows необходимо проходить процедуру активации.

В то же время существуют так называемые корпоративные лицензии, позволяющие быстро установить Windows на большое число компьютеров, не затрачивая время на активацию. Microsoft предпочла предоставить своим крупным клиентам такую возможность, хотя было очевидно, что утечка регистрационного номера всего одной корпоративной лицензии приведет к появлению огромного числа нелегальных, но полностью работоспособных копий Windows (что на практике и случилось с Windows XP, причем пиратские версии XP с регистрационным номером от корпоративной лицензии появились в Интернете ранее чем за месяц до официального начала продаж этой операционной системы).

Подытоживая все вышеизложенное, можно сказать, что когда защита не является основной задачей, решаемой какой-либо программой, необходимо искать оптимальный сбалансированный вариант, при котором будет обеспечиваться удовлетворительный уровень защиты и не пострадает пользователь.

Глава 4



Методы оценки эффективности защиты

Так как почти всегда можно найти более одного способа решения той или иной задачи, желательно иметь некоторые критерии, позволяющие оценивать и сравнивать между собой разные варианты решений.

4.1. Оценка обычных программ

Когда речь идет о прикладных программах, довольно легко применять такие понятия, как качество, надежность и эффективность. *Все эти категории* несут в себе долю субъективизма, но, выяснив мнение нескольких сотен пользователей, на основании собранных статистических данных можно составить вполне реалистичную картину.

4.1.1. Качество программ

Если результат работы программы соответствует ожиданиям пользователей или даже превосходит их, то программу считают качественной. Критерий качества применяется для сравнения результатов процессов, которые можно выполнить более чем одним способом. Так при умножении двух чисел может получиться только один возможный результат, и если в какой-то программе получается другое значение, то правильно будет охарактеризовать программу не как низкокачественную, а как неверно выполняющую расчеты. Но, например, при решении дифференциальных уравнений можно использовать различные численные методы интегрирования, и за большее или меньшее время получить отличающиеся по значениям результаты. При этом несовпадение результатов (как по значению, так и по затраченному времени) обусловлено не ошибками в программе, а свойствами того или иного метода интегрирования и выбранным значением шага (увеличение шага, как правило, приводит к уменьшению времени вычислений, но снижению точности). Качественной будет считаться та программа, которая в подавляющем большинстве случаев удовлетворяет всем требованиям, предъявляемым

пользователем. И, разумеется, с появлением новых технологий или конкурирующих продуктов критерии качества могут становиться жестче, и программа, считавшаяся качественной на протяжении многих лет, может перестать быть таковой, хотя ее функциональность никак не изменилась.

4.1.2. Надежность программ

Надежность программы проще всего определить как ее устойчивость в работе. Из-за очень высокой сложности современных программ далеко не все из них работают безошибочно. Точнее говоря, редко в какой программе не обнаруживались ошибки после успешного прохождения отладки и тестирования. И во многих программах обнаруженные в процессе эксплуатации ошибки даже не исправляются — их просто переводят в разряд документированных особенностей, и пользователям предлагается использовать обходной путь, приводящий к желаемому результату и не вызывающий ошибки. Некоторые ошибки проявляются очень редко и почти случайным образом, что делает их локализацию и исправление чрезвычайно трудной задачей. Так, например, почти любой пользователь Microsoft Office сталкивался с ситуацией, когда Word закрывался с сообщением об ошибке и результаты работы, выполненной с момента последнего сохранения (или автосохранения), оказывались потерянными. Но условия, при которых Word дает сбой, у каждого пользователя могут быть индивидуальными. Более того, ошибка вполне могла произойти не в самой программе текстового редактора, а в одном из общих компонентов Microsoft Office или Windows, используемых редактором Word.

Можно сказать, что надежность программы характеризует безотказность ее работы во всех необходимых пользователю режимах, и чем выше число обнаруженных отказов за определенный период эксплуатации, тем ниже надежность программы.

4.1.3. Экономическая эффективность программ

Под эффективностью программного продукта предлагается понимать не быстродействие, которое лучше называть производительностью и рассматривать наряду с другими качественными характеристиками, а экономическую полезность. Наверное, большинство разрабатываемого в мире программного обеспечения является коммерческим. Разумеется, существуют программы для проведения научных расчетов или автоматизации деятельности организаций, разрабатываемые не для продажи, а исключительно для решения внутренних задач. Существует и бесплатное программное обеспечение. Но основная масса программ разрабатывается с целью реализации их на рынке и получения прибыли. И экономическая эффективность программ-

ного продукта может быть описана соотношением полученной выгоды и затраченных ресурсов.

4.2. Оценка средств защиты

Теперь рассмотрим, что изменится при использовании таких критериев, как качество, надежность и эффективность в приложении к средствам защиты.

4.2.1. Качество защиты

Проблема оценки качества защиты заключается в том, что отличить качественную защиту от некачественной очень трудно. Чем может руководствоваться потенциальный потребитель при выборе необходимых ему средств защиты? Например рекламными материалами, исходящими от разработчика или продавца. Но можно предположить, что, желая продать программу, разработчик будет всячески расхваливать преимущества своего продукта и ни словом не обмолвится о слабых сторонах. Причем описываемые достоинства могут быть и выдуманными, а о некоторых недостатках разработчик не всегда знает в силу заблуждений или элементарной технической безграмотности.

Удивительная программа eBook Pro

Разработчики программы eBook Pro во всю рекламируют свое детище как "единственный программный продукт во вселенной, способный обеспечить Вашей информации практически 100% защиту от взлома". Поверить рекламе, которая обещает именно то, что хочется получить, очень легко. Правда, вскоре неминуемо выяснится, что, нажав комбинацию клавиш <Ctrl>+<A>, можно выделить весь видимый текст, а затем скопировать его в буфер обмена. Кроме того, незащищенные копии HTML-страниц и картинок остаются после просмотра в директории, хранящей кэшированные файлы Internet Explorer. И наконец, выполнив анализ работы программы, можно будет узнать, что защита заключается в наложении при помощи операции XOR на каждый байт защищаемых данных последовательно всех байтов строки "encrypted" (зашифровано), что эквивалентно наложению однобайтовой константы. При этом не обеспечивается никакой секретности и реальной защиты, т. к. остается возможность извлечения защищенной информации без подбора ключа или иных длительных операций. Скорее всего разработчики программы просто не знали, что любое число последовательных вычислений XOR С константой может быть сведено к одному вычислению XOR.

Дополнительную информацию о свойствах продукта можно получить от других пользователей или самостоятельно разбираясь в особенностях функ-

ционирования интересующей программы. Но при оценке качества работы средств защиты далеко не все аспекты можно увидеть невооруженным глазом. Так, например, в случае с программой eBook Pro обнаружить некоторые недостатки внутреннего устройства, такие как возможность выделения текста и копирования его в буфер обмена, а также незащищенные копии просмотренных страниц па диске, может любой пользователь. Но для того чтобы разобраться непосредственно в функционировании защиты (что за алгоритмы применяются и как именно), у обычного пользователя не хватит знаний. Просто взглянув на защищенный документ и работающую с ним программу, почти никогда нельзя с уверенностью сказать, насколько правильно с точки зрения безопасности организована защита. Единственный способ получить достоверную информацию о качестве защитных механизмов — заказать и оплатить проведение независимой экспертизы. Однако стоимость экспертных услуг с большой вероятностью окажется очень высокой по сравнению со стоимостью программного продукта, и редко какой пользователь станет тратить средства на экспертизу. В результате, оценка качества средств защиты почти всегда основывается на сравнении заявленных разработчиком характеристик и интерфейса, хотя ни то ни другое не отражает реальных свойств защиты.

Результаты официального тестирования ключей HASP

Аппаратные ключи для защиты программного обеспечения от несанкционированного тиражирования HASP (Hardware Against Software Piracy), производимые компанией Aladdin Knowledge Systems, Ltd., наверное, являются наиболее распространенными ключами в России. По результатам тестирования, проведенного Национальной Тестовой Лабораторией США (National Software Testing Labs, NSTL), ключи HASP были названы лучшими 2 раза подряд. Согласно отчету NSTL, датированному январем 1999 года, сравнительное тестирование ключей разных производителей велось по пяти категориям: безопасность, простота использования, совместимость, возможности сетевых ключей и универсальность. Ключи HASP оказались лидерами во всех пяти категориях и, следовательно, стали безусловными победителями.

Казалось бы, такой серьезной организации, как NSTL, можно доверять. Но есть несколько нюансов, ставящих под сомнение истинность вердикта, вынесенного NSTL.

Прежде всего, при тестировании сравнивались ключи только двух семейств: HASP от компании Aladdin и Sentinel от компании Rainbow Technologies Inc. Не исключено, что ключи Rainbow Sentinel являются наиболее значимым конкурентом для Aladdin HASP, но на момент тестирования на рынке были представлены ключи и других производителей, сравнение с которыми не проводилось.

Но, самое главное, на момент опубликования отчета NSTL в Интернете можно было найти достаточное количество статей, руководств и даже исходных текстов программ, в деталях описывающих внутреннее устройство ключей HASP, включая алгоритм вычисления секретной функции HaspCode и быстрого поиска пароля для доступа к ключу. Грубо говоря, существовал хорошо документированный инструментарий, позволяющий при наличии физического доступа к ключу HASP за пару минут получить всю информацию, необходимую для построения полного эмулятора, способного на любой корректный запрос к ключу вычислить ответ, совпадающий с ответом реального ключа.

Эмуляции поддаются и другие ключи, но, например, у ключей SentinelSuperPro секретная функция RNBOsproQuery может эмулироваться только таблично *(см. гл. 9)*, а самая трудная для эмуляции часть ключей HASP — секретная функция HaspCode — оказалась скомпрометирована. По одной версии раскрытие алгоритма HaspCode было произведено исследователем программ, писавшим статьи под псевдонимом "bajunny", по другой — произошла утечка секретной информации из компании Aladdin.

Ключи HASP можно было признать сколь угодно простыми и удобными в использовании, универсальными и совместимыми с существующим оборудованием. Но возможность построения полного эмулятора снижает уровень обеспечиваемой безопасности практически до нуля, и такие ключи можно считать лучшими для чего угодно, но только не для защиты программ.

Стоит отметить, что с появлением ключей семейства HASP4 в арсенал разработчика добавились две секретных функции: HaspEncodeData и HaspDecodeData. Это привело к невозможности полной эмуляции ключей HASP4.

4.2.2. Надежность защиты

При оценке надежности средств защиты также имеются отличия от обычных программ. Дело в том, что обычная программа может считаться надежной, если она устойчиво работает во всех необходимых пользователю режимах. И пользователь, как правило, не ставит себе целью найти ошибку в программе — это не соответствует его потребностям. В случае средств защиты кроме пользователя возникает еще одно действующее лицо — противник.

Противник будет стараться найти какой-нибудь недочет в программе, позволяющий ему проломить защиту. И если пользователя обычной программы можно попросить выполнять операции обходным путем, чтобы избежать возникновения ошибки, то от противника всю информацию об ошибках надо тщательно скрывать, иначе он воспользуется ею для достижения своих целей.

существуют задачи защи
нию методов, не имеющи
нит, сложность взлома защ
го устройства "черного ящ
ммными средствами, то н
ествует. И в этой ситуации
"Формула любви": "Ежел
рать может". И противник
браться в том, что придума

Также многократное использование средств защиты способно повысить выгодность взлома. Чем больше разнообразной информации защищается одним и тем же способом, тем большую выгоду удастся извлечь противнику, найдя уязвимость в средствах защиты. И может наступить момент, когда затраты на взлом с лихвой окупятся, даже если стоимость взлома будет чрезвычайно высока.

Если речь идет о защите программного продукта от несанкционированного копирования (тиражирования), то основная цель защиты не сделать невозможным нелегальное использование программного обеспечения, а повысить прибыли от продаж защищаемой программы. Следовательно, сложность и взломостойкость такой защиты не обязательно должны быть высокими.

WinZip

Одним из самых популярных форматов сжатия данных среди пользователей операционных систем семейства Windows был и остается ZIP, разработанный компанией PKWARE, Inc. Столь широкое распространение этот формат получил совсем не из-за технических особенностей, таких как очень быстрое сжатие или высокая степень упаковки, существуют архивные форматы и поддерживающие их программы, превосходящие ZIP практически по всем характеристикам. Скорее всего, ZIP обязан своей популярностью условно бесплатной программе WinZip. Согласно пресс-релизу от 21 ноября 2000 года, на тот момент только с интернет-сайта Download.com, принадлежащего компании CNET Networks, Inc., WinZip скачали более 27 миллионов раз, а июль 2003 года число скачанных копий превысило 100 миллионов.

WinZip является условно бесплатным (shareware) продуктом. Любой пользователь имеет право установить WinZip себе на компьютер и использовать его бесплатно в течение 30 дней с тестовыми и ознакомительными целями. При этом программа является полностью функциональной, но иногда появляется окно-напоминание с предложением купить WinZip, По истечении тестового периода необходимо либо приобрести лицензию на использование WinZip, либо удалить программу с компьютера. После оплаты стоимости лицензии пользователь получает регистрационный код, соответствующий его имени. После ввода правильного кода в соответствующем окне WinZip программа считается зарегистрированной и перестает беспокоить пользователя предложением совершить покупку.

Алгоритм, используемый в WinZip для проверки соответствия регистрационного кода имени пользователя, много лет назад был раскрыт, и в Интернете можно без труда найти исходные тексты и готовые программы, позволяющие вычислить правильный регистрационный код для произвольного имени. Маловероятно, что в WinZip Computing не знают о существовании генераторов кодов

к их программе, но на протяжении многих версий схема регистрации не менялась и, похоже, меняться не будет. Несмотря на сравнительную простоту получения полностью работоспособной копии WinZip без оплаты стоимости лицензии, ужесточение схемы регистрации вряд ли вызовет резкое увеличение объемов продаж. А вот затраты на обновление регистрационных номеров у всех существующих легальных пользователей могут оказаться совсем не маленькими.

Пример сознательного использования слабых механизмов защиты программ можно усмотреть в действиях компании Microsoft.

Регистрация продуктов Microsoft

Долгие годы продукты компании Microsoft (операционные системы, офисные продукты) не использовали сколько-нибудь стойкой защиты от несанкционированного тиражирования. Имея оригинальный дистрибутив, можно было при желании установить его на любое число компьютеров. И не считалось большим преступлением, хотя и являлось нарушением лицензии, когда купленный какой-нибудь компанией программный продукт устанавливался работником не только на офисный, но и на домашний компьютер, чтобы иметь больше возможностей для изучения и освоения новых функций. Похоже, такая ситуация вполне устраивала Microsoft — популяризация собственного программного обеспечения явно шла на пользу компании. А когда пользователь принимал решение о покупке программного обеспечения для домашнего компьютера, выбор чаще падал на продукты Microsoft, ведь с ними он уже был хорошо знаком.

Когда на рынке операционных систем для персональных компьютеров с процессорами семейства x86 компания Microsoft стала безусловным лидером, она приняла решение об изменении правил игры и ввела активацию программных продуктов. Теперь устанавливать дополнительные копии стало довольно трудно. Но если бы активация была заложена в продукты Microsoft с самого начала, не известно, как сложилась бы судьба Microsoft.

Если же обеспечение защиты является основной функцией коммерческого программного продукта, экономическая эффективность может оцениваться так же, как и для обычных программ: чем больше прибыли приносят продажи программы, тем выше эффективность. Но, к сожалению, очень часто в погоне за прибылью производители не обращают достаточно внимания на обеспечение качества и надежности средств защиты. Ведь сложившаяся практика продаж программного обеспечения почти всегда снимает с разработчика ответственность за ущерб, понесенный потребителем при использовании программы.



Часть II

НЕСКОЛЬКО СЛОВ О КРИПТОЛОГИИ

Эта часть поможет читателям поближе познакомиться с криптографией — наукой, предоставившей инструментарий для защиты сохраняемой и передаваемой информации от любого противника. В следующих четырех главах о криптографии рассказывается без использования сложного математического языка, и при этом подробно рассматриваются распространенные ошибки, допускаемые при реализации средств защиты, использующих криптографию.

Глава 5



Базовые понятия

Практически ни одна современная книга, посвященная информационной безопасности, не обходится без упоминания о криптографии. И на это есть серьезные причины: криптография — один из основных инструментов, использующихся при защите информации.

5.1. Происхождение названий

Термин *шифр* (cipher) происходит от арабского слова "цифра" — арабы первыми стали защищать текст, заменяя буквы цифрами.

Криптография (cryptography) дословно переводится как "тайнопись", искусство тайного письма (от греческих слов kryptos — тайный и grapho — пишу). Потребность в криптографии возникала, когда требовалось передавать сообщения таким образом, чтобы их не мог прочитать противник. В историю вошло множество шифров, изобретенных и применявшихся в разные века, в том числе и до нашей эры.

Параллельно с методами шифрования разрабатывались и методы взлома шифров. Исследованием криптографических алгоритмов с целью оценки их стойкости и поиска слабых мест занимается *криптоанализ* (cryptanalysis). Традиционно криптоанализ применялся для чтения перехваченных сообщений без знания ключа или даже метода шифрования. Криптография и криптоанализ являются двумя базовыми составляющими одной науки — *криптологии* (cryptology).

Существует также раздел информационной безопасности, по наименованию созвучный с криптографией — *стеганография* (steganography). Его название происходит от греческих слов stege — крыша и grapho — пишу. Стеганография занимается вопросами скрытной передачи информации, когда ставится задача предотвратить раскрытие противником не только содержимого сообщения, но даже и самого факта, что сообщение было отправлено. Стеганография может

использовать элементы криптографии, но является совершенно отдельным от криптологии направлением и в данной части книги подробно не рассматривается.

5.2. Криптография и наука

Долгое время криптография была больше искусством, чем наукой. Создатели шифров, придумывая алгоритмы преобразования, действовали во многом "на удачу", т. к. в их распоряжении не было подходящей математической теории, способной формализовать криптографические операции и перевести их на язык науки.

Первой работой, радикально изменившей такое положение вещей, принято считать статью американского инженера и математика Клода Шеннона (Claud Shannon) "Теория связи в секретных системах" ("The Communication Theory of Secrecy Systems"), опубликованную в 1949 году в журнале Bell System Technical Journal. Содержимое этой статьи основано на секретном докладе "Математическая теория криптографии", датированном 1 сентября 1945 года. Разумеется, статья была опубликована только после того, как доклад оказался рассекречен.

Статья Шеннона сразу перевела криптографию в разряд точных наук, фактически сделав ее разделом математики. А этап развития криптографии и криптоанализа до 1949 года теперь иногда называют донаучной криптологией.

Кстати, примечательно, что первая программируемая вычислительная машина, носившая имя "Colossus", была создана в Англии в 1943 году. Разработчиками машины были Макс Ньюмен (Max Newman) и Томми Флауэрс (Tommy Flowers). В работах активное участие принимал английский математик Алан Тьюринг (Alan Turing). Вычислительная машина предназначалась для взлома шифра "Enigma", использовавшегося Германией во время второй мировой войны. Таким образом, можно считать, что информатика и вычислительная техника появились благодаря потребностям криптоанализа.

5.3. Терминология

5.3.1. Участники взаимодействия

При любом информационном обмене существует *отправитель* сообщения (sender) и его *получатель* (recipient). Частным случаем этой схемы является ситуация, когда отправитель и получатель — одно и то же лицо, а сообщение передается не в пространстве, а во времени. Именно так может быть

описан процесс хранения информации на внешнем носителе или в памяти компьютера.

Зачастую отправитель желает, чтобы на всем пути следования содержимое сообщения сохранялось в тайне, т. е. чтобы *злоумышленник* (intruder), перехвативший сообщение, не смог понять его смысл. Также в некоторых случаях у злоумышленника есть возможность воздействовать на содержимое сообшений (изменять, удалять, создавать новые сообщения). Подразумевается, что в распоряжении злоумышленника находятся все существующие на настоящий момент технические средства, которые могут помочь в решении его задач.

5.3.2. Объекты и операции

Исходное, незашифрованное сообщение называется *открытым текстом* (plain text). Зашифрованное сообщение называется *шифртекстом* (ciphertext).

Процесс преобразования открытого текста в шифртекст называется *зашифрованием* (enciphering), а обратный процесс — *расшифрованием* (deciphering). Термин *шифрование* (без приставок) в русскоязычной литературе обычно обозначает и зашифрование, и расшифрование.

Зашифрование и расшифрование выполняются в соответствии с *криптографическим алгоритмом* (cryptographic algorithm). Как правило, криптографический алгоритм содержит сменный элемент — *криптографический ключ* (cryptographic key), позволяющий выбрать одно конкретное преобразование из множества преобразований, реализуемых данным алгоритмом.

Существует два основных типа криптографических алгоритмов: *симметричные,* для которых ключ расшифрования совпадает с ключом зашифрования или может быть легко из него получен, и *асимметричные,* использующие для зашифрования и расшифрования два разных ключа. Асимметричные алгоритмы также называют алгоритмами с открытым ключом, и их история начинается с 1975 года, в то время как симметричные алгоритмы использовались многие тысячелетия.

Симметричные алгоритмы можно разделить на две категории. К первой категории относятся алгоритмы, которые обрабатывают шифруемые данные побитово (или посимвольно), и такие алгоритмы называют *потоковыми шифрами*. Ко второй категории относят алгоритмы, производящие операции над группами битов. Такие группы битов называют *блоками*, а алгоритмы — *блочными шифрами*.

Получение открытого текста из шифртекста без знания правильного ключа и/или всех деталей алгоритма является основной задачей *криптоанализа* и называется *дешифрованием*. Попытка криптоанализа называется *атакой*.

Раскрытие ключа шифрования без привлечения методов криптоанализа называется *компрометацией*.

К криптографическим функциям, кроме алгоритмов зашифрования и расшифрования, относят и некоторые другие операции. Так, например, *криптографические хэш-функции* (cryptographic hash-functions) применяются для вычисления значения *хэша* (hash value), называемого еще *дайджестом* сообщения (message digest). Также существуют *криптографические генераторы псевдослучайных чисел* (random number generator).

5.4. Криптографические примитивы

5.4.1. Алгоритмы шифрования

Основная задача криптографии — обеспечение секретности — реализуется при помощи алгоритмов шифрования. Эти алгоритмы, по определению, являются обратимыми, т. к. в противном случае восстановить зашифрованные данные будет не всегда возможно.

Любой алгоритм шифрования, называемый также шифром, представляет собой две связанных математических функции, используемых для прямого и обратного преобразования информации (зашифрования и расшифрования). В'некоторых алгоритмах зашифрование и расшифрование могут выполняться одной и той же функцией.

Раньше защита, обеспечиваемая шифром, часто основывалась на секретности самого алгоритма шифрования. Криптографические алгоритмы, требующие сохранения в тайне последовательности преобразования данных, называются *ограниченными* и в настоящее время практически не находят применения — использование такого алгоритма большим количеством участников информационного обмена затрудняет обеспечение его секретности. А если один из членов рабочей группы, защищавшей внутреннюю информацию ограниченным алгоритмом, решает покинуть группу, то всем оставшимся участникам придется, во избежание возможной утечки информации, переходить на использование другого алгоритма.

Еще одна проблема, связанная с применением ограниченных алгоритмов, заключается в том, что каждая группа пользователей должна применять свой уникальный, никому больше не известный алгоритм. Следовательно, алгоритм должен быть разработан внутри этой группы. А для разработки хорошего алгоритма шифрования необходимы весьма глубокие познания в криптографии, которые есть далеко не у каждого человека.

В современных шифрах применяют другой подход, определяемый принципом Керкхоффса (Kerckhoffs). Согласно этому принципу в криптосистеме

используется сменный элемент, называемый ключом, и секретность шифра обеспечивается секретностью ключа шифрования, а не секретностью алгоритма. Таким образом, открытое опубликование всех деталей реализации криптографического алгоритма не должно снижать надежность шифра, если ключ шифрования сохраняется в секрете. Кроме того, смена ключа выполняется гораздо проще, чем смена алгоритма, особенно если шифрование реализовано аппаратно.

Хороший алгоритм шифрования имеет следующие статистические характеристики:

- отсутствие статистической зависимости между открытым текстом и шифртекстом;
- шифртекст по статистическим характеристикам не отличим от истинно случайной последовательности;
- изменение любого бита в ключе шифрования при неизменном открытом тексте приводит к изменению примерно 50% бит шифртекста (для симметричных алгоритмов);
- изменение любого бита в блоке открытого текста при неизменном ключе шифрования приводит к изменению примерно 50% бит шифртекста (для блочных алгоритмов).

5.4.2. Криптографические хэш-функции

Криптографические хэш-функции призваны преобразовать входную последовательность произвольного размера в выходное значение фиксированной длины. Термин "хэш-функция" используется также для обозначения функции отображения при доступе к хэш-таблицам — структурам данных, используемых во многих алгоритмах. У таких функций много свойств, делающих их схожими с криптографическими хэш-функциями, но это разные вещи, и ни в коем случае не стоит путать хэш-функции для хэш-таблиц с криптографическими хэш-функциями. В этой книге рассказывается только о криптографических хэш-функциях.

Криптографические хэш-функции применяются в криптографии повсеместно: в протоколах аутентификации, цифровой подписи, в генераторах псевдослучайных последовательностей и т. д.

Хорошая хэш-функция равномерно и случайно отображает множество всех возможных входных сообщений во множество результирующих хэшей.

Криптографическая хэш-функция должны быть однонаправленной. То есть, зная значения хэша, злоумышленник не должен иметь эффективной возможности найти исходное сообщение. Более того, не должно быть эффективного способа найти любое сообщение, вычисление хэш-функции от ко-

торого даст требуемое значение хэша (хотя таких сообщений бесконечно много, т. к. количество разных выходных значений определяется размером хэша, а множество входных сообщений безгранично).

Также хорошая криптографическая хэш-функция не должна позволять злоумышленнику подобрать два сообщения, для которых значения хэшей будут совпадать.

5.4.3. Криптографические генераторы псевдослучайных чисел

Случайные числа требуются в криптографии очень часто. Симметричный ключ шифрования должен выбираться случайным образом. При генерации ключей для асимметричной криптосистемы необходимо иметь достаточно большой объем случайных данных. Корректная реализация асимметричных криптоалгоритмов, например RSA, требует добавлять к каждой порции открытого текста несколько случайных байт.

Однако в компьютере, в общем случае, нет хорошего источника случайности, способного выдавать значительные объемы истинно случайных данных. Поэтому в криптографии находят широкое применение генераторы псевдослучайных чисел.

Псевдослучайные данные совсем не то же самое, что истинно случайные. Генератор псевдослучайных чисел использует детерминированный ачгоритм и выдает последовательность значений, зависящую от начального значения (seed value), загруженного в генератор. Зная начальное значение, легко повторить последовательность, выдаваемую генератором.

Большинство языков программирования содержат функции для генерации псевдослучайных чисел. Но эти функции в подавляющем большинстве не удовлетворяют жестким требованиям, которые предъявляет криптография к генераторам псевдослучайных чисел:

- последовательность, выдаваемая алгоритмом генерации псевдослучайных чисел, должна иметь как можно больший период;
- зная любой фрагмент последовательности, выдаваемой генератором, злоумышленник не должен иметь эффективной возможности найти начальное значение, загруженное в генератор;
- зная любой фрагмент последовательности, выдаваемой генератором, злоумышленник не должен иметь возможности получить достоверную информацию о предыдущих или последующих элементах последовательности.

Такие функции генерации псевдослучайных чисел, как, например, rand из стандартной библиотеки языка С, не удовлетворяют ни одному из перечис-

ленных требований. Поэтому не стоит пользоваться для защиты информации генераторами, встроенными в языки программирования, если достоверно не известна их криптографическая стойкость.

5.5. Модели основных криптоаналитических атак

При поиске ключа, необходимого для расшифровки перехваченного сообщения, в распоряжении криптоаналитика всегда находится метод полного перебора ключевого пространства. Поэтому объем ключевого пространства, используемого в криптосистеме, должен быть настолько велик, чтобы в ближайшем (или отдаленном, в зависимости от ценности зашифрованной информации) будущем полный перебор не успел бы завершиться.

Хороший алгоритм шифрования должно быть невозможно вскрыть более эффективным методом, чем полный перебор ключевого пространства.

При оценке стойкости того или иного алгоритма шифрования рассматривают несколько наиболее распространенных моделей криптоаналитических атак.

5.5.1. Атака на основе только шифртекста

При выполнении этой атаки криптоаналитику доступно некоторое количество шифртекстов, являющихся результатом применения одного алгоритма шифрования.

Задача криптоаналитика заключается в том, чтобы найти как можно больше открытых текстов, соответствующих имеющимся шифртекстам, а еще лучше — определить ключ, использованный при зашифровании.

Входные данные для атаки на основе только шифртекста могут быть получены простым перехватом зашифрованных сообщений, что при передаче по открытым каналам связи сравнительно легко реализуемо.

Данная атака является самой слабой и неудобной для криптоаналитика.

5.5.2. Атака на основе открытого текста

При выполнении этой атаки криптоаналитик имеет доступ не только к шифртекстам, но и к соответствующим им открытым текстам.

Задача криптоаналитика сводится к нахождению ключа шифрования, использованного для имеющихся пар текстов, или построению алгоритма, позволяющего расшифровывать любые сообщения, зашифрованные на этом ключе.

Открытые тексты, необходимые для данной атаки, могут быть получены из разных источников. Например, если известно, что передается зашифрованный файл с определенным именем, то по расширению часто можно сделать предположение о содержимом определенных фрагментов файла, например заголовка.

Данная атака сильнее, чем атака на основе только шифртекста.

5.5.3. Атака на основе подобранного открытого текста

В данном случае в распоряжении криптоаналитика также есть некоторое число шифртекстов и соответствующих им открытых текстов. Но, кроме того, криптоаналитик имеет возможность выбрать несколько произвольных открытых текстов и получить соответствующие им шифртексты.

Задача криптоаналитика точно такая же, что и при атаке на основе открытого текста: определить использованный ключ шифрования или найти иной способ расшифровывать сообщения, зашифрованные на том же ключе.

Получить шифртекст, соответствующий заданному открытому тексту, иногда можно, например, создав поддельное незашифрованное сообщение от имени одного из пользователей, обычно использующих шифрование. При совпадении некоторых факторов на такое сообщение может быть создан зашифрованный ответ, цитирующий исходное сообщение.

У криптоаналитика при реализации атаки на основе подобранного открытого текста появляется возможность выбирать блоки открытого текста, что может, в свою очередь, дать дополнительную информацию о ключе шифрования.

5.5.4. Атака на основе адаптивно подобранного открытого текста

Этот вариант является расширением атаки на основе подобранного открытого текста. Отличие заключается в том, что, получив шифртекст, соответствующий выбранному открытому тексту, криптоаналитик может принять решение, какой открытый текст он хочет зашифровать в следующий раз.

Атака на основе адаптивно подобранного открытого текста может применяться, когда у криптоаналитика есть доступ к шифрующему устройству (например к смарт-карте), реализующему определенный алгоритм шифрования по ключу, недоступному для считывания.

Адаптивность (обратная связь) данной атаки дает преимущество перед простой атакой на основе подобранного открытого текста, где все открытые тексты выбирались до начала атаки.

5.6. Анализ стойкости криптографических примитивов

Несмотря на многовековую историю криптографии и криптоанализа, до сих пор не существует математического аппарата, позволяющего доказать, что ключ шифрования определенного алгоритма невозможно найти более эффективно, чем полным перебором ключевого пространства. Скорее всего, такой математический аппарат не будет разработан и в обозримом будущем. Однако прежде чем использовать любой криптографический алгоритм, необходимо получить подтверждение его надежности.

Алгоритмы шифрования с открытым ключом, как правило, позволяют свести задачу взлома шифра к хорошо известной математической проблеме, такой как разложение большого числа на простые сомножители или вычисление дискретного логарифма в конечном поле. Пока математическая проблема не имеет эффективного решения, алгоритм будет оставаться стойким. Как только эффективное решение будет найдено, стойкость всех криптографических алгоритмов и протоколов, использующих данную математическую проблему, резко снизится. Так что разработчикам и пользователям криптосистем, основанных на математической проблеме, остается лишь надеяться, что эффективное решение не существует или никогда не будет найдено. К их счастью, значительных предпосылок к близкому прорыву в этих областях математики пока нет.

Для симметричной криптографии автор алгоритма может привести соображения, которыми он руководствовался при разработке шифра, но этого явно недостаточно. Требуется некая процедура, если не гарантирующая стойкость, то хотя бы дающая высокую степень уверенности, что алгоритм не будет взломан злоумышленником.

На настоящий момент основным методом проверки криптографической стойкости алгоритмов является экспертная оценка. Новый алгоритм открыто публикуется, и все желающие получают возможность попытаться найти в нем слабые места. Если кому-то из криптоаналитиков удается обнаружить серьезные недостатки, алгоритм отправляется в мусорную корзину. Если же на протяжении значительного периода времени (обычно нескольких лет) никому не удалось отыскать уязвимости в алгоритме, то он может занять почетное место среди других алгоритмов, рекомендуемых к применению на практике. Именно так проводятся сейчас конкурсы на выбор алгоритма для национального стандарта шифрования.

Проверить надежность нового алгоритма без привлечения криптографической общественности под силу разве что таким организациям, как АНБ. А любой другой разработчик, желающий сохранить алгоритм в тайне, может

оказаться в ситуации, когда через некоторое время используемый алгоритм перестает быть секретным и вскоре появляется эффективный метод его вскрытия. Именно поэтому современная криптография, в большинстве случаев, является открытой — вероятность взлома хорошо исследованного алгоритма значительно ниже, чем алгоритма, державшегося долгое время в секрете.

Алгоритм шифрования А5

В качестве наглядного примера опасности, связанной с засекречиванием деталей реализации алгоритмов шифрования, можно привести историю поточного шифра А5, применяемого для шифрования сеансов телефонной связи между трубкой абонента и базовой станцией в европейской системе мобильной цифровой связи GSM (Group Special Mobile).

Шифр А5 был разработан в 1989 году и существует в двух версиях: А5/1 — "сильная" версия шифра, разрешенная к применению только в некоторых странах, и А5/2 — "ослабленная" версия, разрешенная к свободному применению. В 1989 году широкая публикация алгоритмов не была распространенным подходом, и детали построения А5 оказались засекречены.

Но как бы строго ни контролировались коммерческие секреты, широкое распространение продукции рано или поздно приводит к утечкам информации. В случае с GSM утечки начались в начале 90-х годов. Британская телефонная компания передала всю документацию Брэдфордскому университету, не потребовав от него подписать соглашение о неразглашении. Часть информации попала в Интернет, а к 1994 году основные детали алгоритма А5 стали общедоступны. В конце концов, кембриджские ученые Майк Роэ (Mike Roe) и Росс Андерсон (Ross Anderson) опубликовали в Интернете примерную схему алгоритма.

В начале 1999 года в ассоциации разработчиков смарт-карт (Smart-Card Developer Association, SDA) были полностью восстановлены и проверены на реальных тестовых векторах схемы алгоритмов А5/1 и А5/2. Почти сразу после *этого* была предложена атака, позволяющая вскрывать шифр А5/2 на персональном компьютере всего за 15 миллисекунд.

В декабре 1999 года израильскими математиками Ади Шамиром (Adi Shamir) и Алексом Бирюковым (Alex Biryukov) была опубликована еще одна работа, в которой описан нетривиальный, но по теоретическим расчетам очень эффективный метод вскрытия алгоритма А5/1. Этот метод требует 2^{48} предварительных вычислений и позволяет находить ключ за 1 секунду на персональном компьютере, имеющем 128 Мбайт оперативной памяти и 150 Гбайт дискового пространства, путем анализа выхода алгоритма в течение первых двух минут телефонного разговора.

Однако интуитивно понятно, что отсутствие успешных результатов криптоанализа конкретного алгоритма еше не гарантирует, что эти результаты не появятся в будущем. Работы по усовершенствованию методов криптоанализа ведутся постоянно, и нет никакой гарантии, что не удастся найти эффективные методы взлома существующих шифров.

Экспертная оценка применяется аналогичным образом и для проверки криптографической стойкости хэш-функций и генераторов псевдослучайных чисел.

Глава 6



Криптография для нематематиков

Многие люди, пытающиеся заняться глубоким изучением криптографии, могут столкнуться с тем, что эта задача им не под силу. Уж слишком серьезная математическая подготовка требуется для того, чтобы детально понимать, как строить надежные алгоритмы и как выполнять их криптоанализ. Но разработчику программ, связанных с защитой информации, не обязательно знать все о самих алгоритмах — достаточно уметь их правильно применять, не оставляя противнику возможности для атаки.

6.1. Открытая криптография в России

Каких-нибудь 15 лет назад криптография в России (тогда еще Союзе Советских Социалистических Республик) была чем-то вроде технологии производства оружия — существование криптографии не являлось тайной, и почти в любом кинофильме про разведчиков (или шпионов, если фильм был иностранного производства) фигурировал человек, зашифровывающий или расшифровывающий секретные сообщения. Но все, что было связано с реальной криптографией, находилось в области действия военных или спецслужб, т. е. под контролем государства. Следовательно, в книжных магазинах нельзя было найти популярных изданий по криптографии, а в открытых библиотеках не было соответствующих научных работ — криптография была закрытой.

Правда, 2 июня 1989 года Постановлением Государственного комитета СССР по стандартам № 1409 был утвержден и введен в действие (с 1 июля 1990 года) ГОСТ 28147-89 "Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования".

Общее *описание ГОСТ 28147-89*

Настоящий стандарт устанавливает единый алгоритм криптографического преобразования для систем обработки информации в сетях электронных вычислительных

машин (ЭВМ), отдельных вычислительных комплексах и ЭВМ, который определяет правила шифрования данных и выработки имитовставки.

Алгоритм криптографического преобразования предназначен для аппаратной или программной реализации, удовлетворяет криптографическим требованиям и по своим возможностям не накладывает ограничений на степень секретности защищаемой информации.

Стандарт обязателен для организаций, предприятий и учреждений, применяющих криптографическую защиту данных, хранимых и передаваемых в сетях ЭВМ, в отдельных вычислительных комплексах или ЭВМ.

Стоит напомнить, что в США первый стандарт шифрования DES был опубликован и вступил в силу в 1977 году, на 13 лет раньше, чем ГОСТ 28147-89.

Трудно сказать точно, в какой именно момент в криптографии произошел информационный прорыв. Скорее всего, когда в России доступ в Интернет появился у значительного числа обычных пользователей, т. е. в первой половине 90-х годов XX века. В Интернете обнаружились ресурсы, содержащие огромные объемы информации по криптологии: описания криптографических алгоритмов и протоколов, статьи, по криптоанализу, исходные тексты и т. д. В таких условиях криптография как наука не могла больше оставаться секретной, к тому же развитие средств коммуникации в России выявило потребность в криптографии не только для спецслужб, но и для коммерческих структур.

Государство не осталось безучастным к возникшей открытости криптографии, и 3 апреля 1995 года под давлением ФАПСИ (Федерального агентства правительственной связи и информации при Президенте Российской Федерации) появился Указ Президента Российской Федерации № 334 "О мерах по соблюдению законности в области разработки, производства, реализации и эксплуатации шифровальных средств, а также предоставления услуг в области шифрования информации". Ниже приводится небольшая выдержка из этого Указа.

Фрагмент Указа № 334 (пункт 4)

В интересах информационной безопасности Российской Федерации и усиления борьбы с организованной преступностью запретить деятельность юридических и физических лиц, связанную с разработкой, производством, реализацией и эксплуатацией шифровальных средств, а также защищенных технических средств хранения, обработки и передачи информации, предоставлением услуг в области шифрования информации, без лицензий, выданных Федеральным агентством правительственной связи и информации при Президенте Россий-

ской Федерации в соответствии с Законом Российской Федерации "О федеральных органах правительственной связи и информации".

Указ № 334 недвусмысленно запрещает использование криптографии без лицензии даже физическими лицами. Основной аргумент ФАПСИ, касающийся положений Указа, сводился к тому, что нельзя допустить попадания средств защиты информации в руки террористов. Однако программное обеспечение, предназначенное для криптографической защиты информации, в отличие от специализированных средств шифрования связи, может легко быть получено через Интернет любым человеком или организацией.

По мнению общественной некоммерческой ассоциации "РусКрипто", несовершенство законов, принятых в первую очередь под давлением ФАПСИ, реально сдерживает развитие не только криптографии как науки, но и целого сегмента рынка Российской экономики, связанного с системами защиты информации. И исправить сложившуюся ситуацию можно только после внесения смягчающих поправок в законодательство.

Таким образом, правовое поле в вопросах, касающихся разработки и применения криптографических средств, все еще находится в стадии формирования, и законодателям предстоит большая работа в данном направлении.

Кстати, указом Президента России от 11 марта 2003 года ФАПСИ было расформировано, а его функции распределены между Федеральной службой безопасности и Министерством обороны.

6.2. Литература по криптологии

Первые работы, связанные с криптографией и криптоанализом, появились очень давно. Так, например, известно, что Аристотелю, жившему в 384—322 гг. до н. э., принадлежит способ взлома шифра, использовавшегося греками в V—IV вв. до н. э.

Очень важный труд, имеющий отношение к криптоанализу, был написан в IX веке одним из крупнейших ученых арабского мира, носившим имя Абу Юсуф Якуб ибн-Исхак ибн-Ас-Сабах ибн-Умран ибн-Исмалил Аль Кинди, более известный на западе как Алькиндус. В его работе впервые описывалось применение частотного анализа для взлома шифров простой замены.

До первой мировой войны информация о новейших достижениях в криптологии периодически появлялась в открытой печати. В 1918 году увидела свет монография Вильяма Фридмана (William F. Friedman) "Индекс совпадения и его применение в криптографии" ("Index of Coincidence and Its Applications in Cryptography"). Это была одна из самых значительных работ XX века в области криптоанализа.

После первой мировой войны открытых новаторских работ по криптографии почти не публиковалось. Следующим значительным событием стала уже упомянутая статья Клода Шеннона "Теория связи в секретных системах" *(см. гл. 5),* появившаяся в 1949 году.

В 1967 году вышла книга Дэвида Кана (David Kahn) "Взломщики кодов" ("The Codebreakers: The Story of Secret Writing"), которая не была посвящена научной стороне криптографии, но содержала огромный исторический материал. В книге описывались имевшие место случаи успешного использования криптоанализа, включая факты, которые правительство США все еще считало секретными.

Значительный импульс широкому развитию криптографии дала опубликованная в 1976 году статья Уитфилда Диффи (Whitfield Diffie) и Мартина Хеллмана (Martin Hellman) "Новые направления в криптографии" ("New Directions in Cryptography"), положившая начало криптографии с открытым ключом.

Статья Диффи и Хеллмана подтолкнула к занятиям криптографией большое количество людей, что привело к значительному росту числа опубликованных разными авторами книг и статей. В настоящее время новые работы появляются чуть ли не ежедневно. Так, например, список литературы, приведенный в русскоязычном издании книги Брюса Шнайера (Bruce Schneier) "Прикладная криптография" ("Applied Cryptography"), занимает 56 страниц и содержит ссылки на 1653 работы. *J*

Довольно интересно сложилась ситуация с литературой по криптографии в России. С того момента, как криптография оказалось рассекречена, начали появляться открытые публикации, в разных издательствах вышло несколько десятков книг, у которых в названии присутствует слово с корнем "крипто". Информационное наполнение всех этих изданий можно разделить на четыре основных категории.

К первой категории относятся история криптографии и описание древних методов шифрования. Для общего развития, конечно, полезно уметь взламывать шифр Цезаря, но в решении реальных задач подобные знания помогают очень редко.

Ко второй категории относится высоконаучная литература по криптографии. Она может показаться увлекательной людям, у которых не оставляет ощущения недопонимания происходящего фраза "канторовское множество возрастает на мере Лебега нуль" (впрочем, не имеющая отношения к криптографии). Однако для большинства людей, не имеющих глубокого математического образования, понять то, что относится к научной криптографии, почти нереально: все-таки криптография нашего времени — это чистая математика. К счастью, данная категория литературы нужна, в основном, раз-

работникам криптографических алгоритмов и криптоаналитикам и почти бесполезна для тех, кто использует готовые алгоритмы и протоколы.

В третью категорию попадают описания общих понятий криптографии и спецификации различных криптографических алгоритмов и протоколов. Используя подобную информацию, любой программист сможет реализовать симметричное и асимметричное шифрование, генерацию и проверку цифровой подписи, а также решить и другие задачи, требующие применения криптографии. Правда, знание всех деталей используемых алгоритмов и безошибочная их реализация на языке программирования еще не являются гарантией того, что полученная система будет стойкой ко всем известным атакам.

К последней, четвертой категории относятся рекомендации по правильному применению криптографии. Именно эта информация является жизненно необходимой при создании реальных систем, использующих криптографию. Однако редкая книга акцентирует внимание читателя на том, что нужно делать обязательно, а чего никогда нельзя допускать.

Из книг по криптографии, изданных на русском языке, в первую очередь стоит порекомендовать уже упомянутую "Прикладную криптографию" Брюса Шнайера. В аннотации к русскому изданию сказано, что это самая читаемая книга по криптографии в мире. Ее первое издание вышло в США в 1994 году, а второе, исправленное — в 1996 году. Русскоязычная версия, официально появившаяся только в 2002 году, основана на втором издании и не отражает событий, произошедших за последние годы. Однако информация, собранная на восьми с лишним сотнях страниц, не становится от этого менее актуальной и полезной. А, по субъективным ощущениям, большинство изданных в России книг, содержащих описания криптографических алгоритмов, заимствуют их именно из "Прикладной криптографии" (и эта книга — не исключение). Но, к сожалению, иногда заимствование производится в весьма значительных объемах и без ссылки на первоисточник.

Несмотря на все бесспорные достоинства, к русскому изданию "Прикладной криптографии" стоит относиться с некоторой осторожностью. Дело в том, что примерно в середине 2001 года в Интернете появилась электронная версия "Прикладной криптографии" на русском языке, не содержащая указаний на авторов перевода. Точность русскоязычной версии, похоже, была не очень высока, а вторая половина девятой и вся десятая глава вообще не были переведены. Скорее всего, бумажное издание 2002 года основывается именно на упомянутой электронной версии. Во всяком случае, по словам научно-технического редактора перевода Павла Семьянова, при редактировании было исправлено огромное количество ошибок, опечаток, неточностей, ошибок переводчика, неверных терминов и т. п., и Семьянов не рекомендует использовать электронную версию, т. к. она содержит принципиальные, критические ошибки, в том числе в описании алгоритмов и протоколов.

Но даже после редактирования перевод содержит некоторое количество ошибок, унаследованных от электронной версии. Список найденных опечаток доступен по адресу:

http://www.ssl.stu.neva.ru/psw/crypto/appl_rus/errata.html

Существует еще одна книга-справочник по криптографии, имеющая похожее название — "Handbook of Applied Cryptography". Она была впервые издана в 1996 году и не переводилась на русский язык. Полная электронная версия книги доступна по адресу:

http://www.cacr.math.uwaterloo.ca/hac/

6.3. Что нужно знать программисту

Программист должен не только уметь закодировать криптографический алгоритм, но и понимать основные свойства криптографических примитивов, с которыми ему придется работать. Применение криптографии вслепую, необдуманно, порождает большую вероятность того, что полученная система окажется уязвимой.

Программист должен понимать, что при кодировании защиты необходимо рассчитывать не на обычного пользователя, которому лицензионным соглашением можно запретить выполнять определенные операции, а на умного и расчетливого профессионала, имеющего самые серьезные намерения по обходу или взлому защиты. И лучше всего исходить из предположения, что злоумышленнику доступны все существующие на настоящий момент знания и технологии, включая даже те, которые не доступны разработчику.

Кроме того, никогда нельзя забывать, что взлом криптографической части защиты может остаться незамеченным. Злоумышленник, однажды научившись расшифровывать перехваченные сообщения, может заниматься этим очень долго, практически не опасаясь быть пойманным. Обнаружить пассивное воздействие, например чтение пакетов, передающихся по сети, практически невозможно.

И, разумеется, при проектировании защиты нужно иметь в виду, что стойкость всей системы определяется стойкостью самого слабого ее звена.

Теперь рассмотрим основные свойства базовых криптографических примитивов.

6.3.1. Блочные шифры

Блочные алгоритмы шифрования применяются, пожалуй, чаще, чем любые другие шифры. Блочный шифр выполняет операции над блоками — порциями данных фиксированного размера. Обычно размер блока составляет

64 бита (8 байт) или 128 бит (16 байт), но некоторые алгоритмы используют и другие значения. Блочный шифр всегда преобразует определенный блок открытого текста в один и тот же шифртекст независимо от того, какие данные были зашифрованы до этого.

Так как размер шифруемого сообщения не всегда кратен размеру блока, возникает проблема дополнения, имеющая несколько решений. Можно, например, передавать в незашифрованном виде размер полезной части зашифрованных данных и после расшифровки лишние байты просто отбрасывать. На практике же часто применяют другой способ.

Если алгоритм оперирует 8-байтовыми блоками данных, а в последнем блоке, например, только 3 байта полезных данных, то все неиспользуемые байты, кроме самого последнего, можно заполнить любым значением, а в последнем байте записать число, равное количеству неиспользуемых байтов. Тогда, получив и расшифровав все блоки, необходимо отбросить с конца столько байт, сколько указано в последнем байте последнего блока. Правда, если исходное сообщение имело длину, кратную размеру блока, то возникает некоторая сложность, т. к. требуется добавить 0 байт, и последний байт должен содержать число байт дополнения. Для разрешения этой проблемы при зашифровании добавляется новый блок, последний байт которого содержит размер блока. Дополнительный блок будет целиком отброшен при расшифровании.

Блочные алгоритмы шифрования могут быть использованы в нескольких режимах, например:

- режим простой замены (Electronic CodeBook, ECB);
- с зацеплением блоков шифртекста (Cipher Block Chaining, CBC);
- с обратной связью по шифртексту (Cipher FeedBack, CFB);
- с обратной связью по выходу (Output FeedBack, OFB);
- по счетчику (Counter);
- с зацеплением блоков открытого текста (Plaintext Block Chaining, PBC);
- с обратной связью по открытому тексту (Plaintext FeedBack, PFB);
- с усиленным сцеплением блоков шифртекста (различные модификации режима CBC);
- с обратной связью по выходу и нелинейной функцией (Output FeedBack with Nonlinear Function, OFBNLF);
- по счетчику с нелинейной функцией (Counter with Nonlinear Function, CNLF).

Этот список далеко не полный и может быть расширен чуть ли не до бесконечности.

Каждый режим имеет свои особенности: он может требовать те или иные дополнительные операции, быть устойчивым или, наоборот, неустойчивым к определенным атакам, лучше или хуже восстанавливаться при сбоях и т. д. Так, например, режим простой замены рекомендуется использовать только для шифрования коротких сообщений, содержащих данные, близкие к случайным (такие как ключи шифрования). Детальное описание режимов применения блочных шифров можно найти в различных изданиях, посвященных криптографии.

6.3.2. Потоковые шифры

Потоковый шифр выполняет операции над битами или символами (например 8-, 16- или 32-битовыми). Потоковый шифр преобразует один и тот же символ открытого текста в разные символы шифртекста, например в зависимости от того, сколько и каких символов было обработано ранее.

Во многих потоковых шифрах зашифрование производится следующим образом. Генератор гаммы, основанный на генераторе псевдослучайных чисел, выдает последовательность битов (гамму). Эта гамма накладывается на открытый текст с помощью побитовой операции XOR. В результате получается шифртекст. Для расшифрования необходимо выполнить в точности ту же процедуру, только наложить гамму, полученную с использованием идентичного генератора с точно таким же начальным состоянием, на шифртекст.

Таким образом, стойкость алгоритма зависит исключительно от характеристик гаммы, выдаваемой генератором. Если гамма состоит из одних нулей (вырожденный случай), то данные при шифровании вообще не изменяются. Если гамма имеет короткий период (например 32 бита), то шифрование сводится к операции XOR С 32-битовой константой. Если же гамма представляет собой случайный набор битов, не подчиняющийся никакой закономерности, получается аналог одноразового шифровального блокнота, который обеспечивает абсолютную защиту. Разумеется, детерминированный алгоритм, используемый в генераторе гаммы, не может выдавать истинно случайную последовательность. Если последовательность не удастся повторить, то не удастся и расшифровать сообщение.

Если два сообщения были зашифрованы с использованием одной и той же гаммы и для одного из сообщений (более длинного) удалось каким-нибудь образом получить открытый текст, то легко получить открытый текст и для другого сообщения. Применив операцию XOR К открытому тексту и шифртексту первого сообщения, мы получим фрагмент гаммы. А наложив гамму на шифртекст второго сообщения, мы получим его открытый текст. Именно поэтому нельзя допускать, чтобы одна и та же гамма использовалась при шифровании двух разных потоков или сообщений.

Если гамма генерируется независимо от содержимого сообщения (как в приведенном ранее примере), то такой потоковый алгоритм называется *синхронным*. Как правило, в синхронных потоковых шифрах ключ шифрования используется для установки начального внутреннего состояния генератора гаммы.

В *самосинхронизирующихся* потоковых шифрах каждый бит гаммы зависит от фиксированного числа предыдущих битов шифртекста.

Более детальное описание достоинств и недостатков потоковых шифров также можно найти в специализированной литературе по криптографии.

6.3.3. Алгоритмы с открытым ключом

Асимметричные алгоритмы подразумевают использование двух математически связанных ключей. Один ключ называют персональным (секретным), и он должен храниться в тайне, а парный ему ключ называют публичным (открытым), и доступ к нему должны иметь все участники информационного обмена. Необходимым требованием для стойкого алгоритма является невозможность эффективного вычисления секретного ключа по открытому ключу.

Стойкие алгоритмы с открытым ключом, как правило, основываются на математических задачах, которые на настоящий момент не имеют эффективного решения. Однако далеко не все стойкие алгоритмы применяются на практике. Некоторые из них требуют очень больших ключей. Например, размер открытого ключа криптосистемы HFE (Hidden Fields Equations) может достигать десятков мегабайт, что затрудняет распределение таких ключей. При использовании некоторых алгоритмов размер шифртекста значительно превышает размер соответствующего ему открытого текста. Не последнюю роль играет и скорость выполнения шифрования — все асимметричные алгоритмы значительно медленнее, чем симметричные.

Алгоритмы с открытым ключом используются для решения двух основных задач: шифрования данных и цифровой подписи, причем многие алгоритмы пригодны для решения только одной из этих задач. Также некоторые алгоритмы позволяют вырабатывать общий сеансовый ключ, который злоумышленник не сможет получить, даже если он перехватит все сообщения между абонентами.

Общей проблемой алгоритмов с открытым ключом является необходимость распространения этих самых открытых ключей. Используя асимметричную криптографию, можно вести защищенный диалог с абонентом, с которым был совершен обмен открытыми ключами. Но в общем случае, не проводя подготовительных действий и не имея общего секрета, невозможно с уверенностью утверждать, что абонент является именно тем, за кого он себя выдает. Для решения этой проблемы используется инфраструктура откры-

тых ключей (Public Key Infrastructure, PKI), которая при помощи иерархии сертификатов позволяет свести доверие абоненту к доверию корневому сертифицирующему центру.

Пожалуй, самым известным на сегодняшний день асимметричным алгоритмом, пригодным как для шифрования, так и для подписи сообщений, является алгоритм RSA, основанный на сложности задачи факторизации (разложения числа на простые сомножители).

6.3.4. Хэш-функции

Хэш-функция должна отображать входные данные произвольного размера в выходной набор бит фиксированного размера. Отображение должно быть равновероятным и случайным.

Похожие требования предъявляются к функциям вычисления контрольных сумм (Cyclical Redundancy Check, CRC), таким как CRC32 или Adler32. Но контрольные суммы призваны, в первую очередь, обнаруживать случайные нарушения целостности. Так что задача подбора двух сообщений, контрольные суммы для которых будут совпадать, или нахождения сообщения с заданным значением контрольной суммы может быть решена эффективно. Например, достаточно откорректировать всего 4 идущих подряд байта, расположенных в любом месте изменяемого сообщения, чтобы значение CRC32 для этого сообщения осталось таким же, как до внесения изменений. Поэтому контрольные суммы (обычно очень простые в реализации) не стоит использовать в криптографии.

Если на выходе хэш-функции, реализующей идеальное случайное равновероятное отображение, получается, например, 128-битовое значение и хэш-функция была вычислена от 2^{128} разных сообщений, это не значит, что каждое из 2^{128} возможных выходных значений получилось ровно один раз. Действительно, предположим, что мы вычислили хэш ровно от половины (2^{127}) входных сообщений, и получили 2^{127} разных выходных значений, т. е. не было ни одного повторения. Учитывая тот факт, что отображение случайно и равновероятно значение хэш-функции от следующего сообщения с вероятностью $^1/_2$ будет совпадать с одним из уже полученных значений. И с каждым новым значением хэша вероятность возникновения коллизии будет только возрастать.

Если результат вычисления хэш-функции без каких-либо изменений и дополнений снова подается на вход той же хэш-функции и так повторяется многократно (например для снижения скорости атаки подбором), мы можем получить вырождение хэша. Вырождение возникает, когда любые входные сообщения отображаются в очень малое множество выходных значений (значительно меньшее, чем 2^{128}) и вероятность подбора двух сообщений с одинаковым значением хэша становится сравнительно большой.

Для того чтобы хэш не вырождался при циклическом вычислении, необходимо на каждом раунде подавать на вход хэш-функции некоторые новые данные, например номер раунда.

6.3.5. Генераторы случайных чисел

Наверное, еще раз стоит повторить, что привычные генераторы псевдослучайных чисел, реализованные в стандартных библиотеках популярных языков программирования, не пригодны для криптографии.

Для того чтобы использовать криптографические генераторы псевдослучайных чисел, их необходимо проинициализировать истинно случайными данными, полученными из физических источников. Популярным способом сбора случайных данных является измерение задержек между нажатиями клавиш на клавиатуре или анализ перемещения указателя мыши, выполняемого пользователем. Однако оба этих способа имеют два серьезных недостатка.

Первый недостаток заключается в том, что невозможно точно сказать, сколько бит действительно случайных данных может быть получено из одной пары нажатий или одного движения мышью. У людей с профессиональными навыками машинописи, как правило, очень высокая ритмичность нажатия клавиш. А случайные данные должны получаться независимо от того, кто сидит за клавиатурой. Да и щелчки при нажатии кнопок могут быть легко записаны злоумышленником на магнитофон, а потом воспроизведены для повторения задержек. С мышью тоже не все понятно — данные, которые получает программа, проходят много инстанций. Мышь передает информацию о перемещении с определенной периодичностью, а не в тот момент, когда датчик перемещения получает информацию о том, что мышь была сдвинута. А драйвер мыши извещает программу о состоянии мыши со своей дискретностью. Все это приводит к тому, что программа получает далеко не полную информацию о том, что произошло с мышью, и основная часть случайных данных, на которые рассчитывала программа, может при определенной комбинации мыши, драйвера и компьютера оказаться совсем не случайной.

А второй недостаток связан с необходимостью присутствия человека для получения случайных данных, что является большой проблемой для серверных приложений.

В качестве претендентов на действительно случайные данные можно рассматривать значение хэша от содержимого экрана и от всех данных, прочитанных с диска с момента загрузки операционной системы. Однако очевидно, что пока не запущено ни одно интерактивное приложение, состояние экрана можно попытаться предугадать, а сразу после получения порции случайных данных повторный запрос может вернуть тот же самый результат.

Иногда в распоряжении программиста оказывается аппаратное устройство, являющееся, согласно прилагаемой документации, генератором истинно случайных чисел. Наличие такого источника случайности очень полезно во многих ситуациях, но таит в себе опасность.

Проблема в том, что по выходу генератора нельзя однозначно определить, действительно ли выдаваемая последовательность случайна. Можно создать такой генератор псевдослучайных чисел, выход которого успешно пройдет все известные статистические тесты на случайность. И все же, выход генератора на самом деле будет определяться детерминированным алгоритмом.

Если злоумышленнику известны детали алгоритма, используемого в аппаратном генераторе случайных чисел (являющихся, на самом деле, псевдослучайными), то злоумышленник может предсказать выход генератора, а значит, и ключи шифрования, если они выбираются по генератору.

Поэтому использовать аппаратные генераторы стоит лишь в тех случаях, когда генератор создан самостоятельно или разработан стороной, которой можно безгранично доверять, и неизменность конструкции подтверждена экспертизой.

6.3.6. Криптографические протоколы

Криптографические алгоритмы почти всегда используются в соответствии с определенным набором правил, называемым протоколом. Если в криптографическом протоколе используется нестойкий алгоритм, то и протокол, наверняка, окажется нестойким. Но использование исключительно стойких алгоритмов еще не гарантирует того, что протокол тоже будет стойким. На практике ошибки в криптографических протоколах обнаруживаются гораздо чаще, чем в криптографических алгоритмах.

6.4. Разработка собственных криптографических алгоритмов

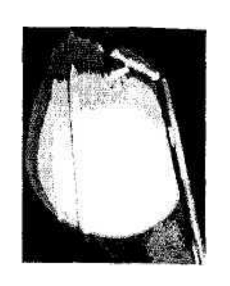
Иногда программист приходит к идее разработать свой собственный криптографический алгоритм или получает аналогичные инструкции от начальства. Разумеется, ничего плохого в такой попытке нет, но практика показывает, что очень редко кто достигает на этом пути положительных результатов.

Дело в том, что для создания действительно хорошего алгоритма мало потребности и желания. Еще требуются глубокие знания, позволяющие проверить стойкость разработанного алгоритма ко всем известным методам криптоанализа. Кроме того, настоятельно рекомендуется открыто опубликовать

алгоритм, чтобы широкая криптографическая общественность попыталась найти в нем слабые места.

В общем, на настоящий момент существует достаточно много самых разнообразных криптографических алгоритмов, почти на любой вкус. Большинство популярных алгоритмов прошли проверку временем и заслужили положительные отзывы криптоаналитиков. И разработка нового алгоритма, наверное, имеет смысл только в одном из двух случаев: если новый алгоритм будет быстрее уже существующих (без потери стойкости) или если он будет более стойким.

Глава 7



Насколько надежны алгоритмы и протоколы

Криптография является существенной и достаточно надежной составной частью практически любой системы защиты информации. Однако иногда защита ослабляется или оказывается взломанной именно из-за проблем в криптографических операциях.

7.1. Слабости в алгоритмах

Криптографические алгоритмы сами по себе проходят многократную экспертную проверку, прежде чем начинают массово применяться на практике. Бывают, конечно, редкие исключения, но они скорее только подтверждают правило.

Одна из криптографических хэш-функций, разработанных Роном Ривестом (Ronald Rivest), называется MD4. Аббревиатура MD расшифровывается как Message Digest (дайджест сообщения). В течение примерно 2-х лет после опубликования спецификации MD4 было представлено как минимум 3 серьезных независимых работы, посвященных криптоанализу хэш-функции MD4. В одной из этих работ описывался взлом последних двух из трех раундов алгоритма обработки данных, а в остальных работах — взлом первых двух раундов. И хотя алгоритм в целом устойчив ко всем этим методам взлома, MD4 не рекомендуется использовать в реальных приложениях.

Не лишен недостатков и потоковый алгоритм шифрования, используемый в архивах формата ZIP. Этот алгоритм был разработан Роджером Шлафлай (Roger Schlafly). Внутреннее состояние шифра определяется тремя 32-битовыми регистрами, инициализируемыми следующими значениями:

```
key0 = 0x12345678;

key1 = 0x23456789;

key2 = 0x34567890;
```

Алгоритм изменения внутреннего состояния может быть представлен следующей функцией на языке С:

```
unsigned char PKZIP_stream_byte (unsigned char pt) {
  unsigned short temp;

  key0 = crc32 (key0, pt) ;
  key1 = (key1 + (key0 & 0xFF)) * 0x08088405 + 1;
  key2 = crc32 (key2, key1 >> 24);
  temp = (key2 & 0xFFFC) | 2;
  return ((temp * (temp ^ 1)) >> 8) & 0xFF;
}
```

Здесь pt (от "plaintext") содержит следующий байт открытого текста, возвращаемое функцией значение представляет собой следующий байт шифртекста, а `crc32` — макрос или функция, принимающая предыдущее значение CRC32 и очередной байт и вычисляющая следующее значение многочлена CRC32, образованного "магическим числом" 0xEDB88320.

Загрузка ключа шифрования (установка состояния внутренних регистров) происходит путем передачи функции `PKZIP_stream_byte` последовательно всех байтов пароля. Возвращаемые значения при этом игнорируются.

В 1994 году Эли Бихэм (Eli Biham) и Пол Кошер (Paul Kocher) опубликовали статью, посвященную атаке на алгоритм шифрования ZIP. Для нахождения ключа шифрования (состояния внутренних регистров после загрузки пароля) достаточно знать 13 последовательных байт открытого текста и примерно 2^{38} раз выполнить действия, представленные в функции `PKZIP_stream_byte`. Если же объем доступного открытого текста превышает 13 байт, трудоемкость атаки значительно снижается. Так, например, наличие 40 байт открытого текста позволяет найти ключ шифрования всего за 2^{34} операций, 110 байт — за 2^{32} операций, а 1000 байт — за 2^{29}.

Несмотря на то, что недостатки этого алгоритма были опубликованы почти 9 лет назад, он до сих пор остается самым часто используемым в архивах формата ZIP. Некоторое время назад в архиваторах PKZIP и WinZip появилась поддержка других, более стойких алгоритмов шифрования, но пока новое шифрование не слишком популярно по нескольким причинам. Во-первых, новые форматы зашифрованных данных в PKZIP и WinZip не совместимы между собой — то, что создано одним архиватором, не может быть прочитано другим (и похоже вообще никаким другим архиватором, в то время как старый формат шифрования поддерживали практически все программы, умеющие работать с архивами ZIP). А во-вторых, компания

PKWARE, создавшая PKZIP, обвиняет авторов WinZip в том, что, реализовав свое шифрование, они нарушают патенты, принадлежащие PKWARE.

7.2. Ошибки в кодировании алгоритмов

Многие криптографические алгоритмы весьма сложны, и при их реализации легко допустить ошибку. Хотя можно допустить ошибку и при реализации сравнительно простых алгоритмов.

В феврале 2001 года в почтовой рассылке cryptography-digest происходило обсуждение ошибки при реализации алгоритма шифрования Alleged RC4 на языке ADA.

История алгоритма шифрования RC4

Потоковый шифр RC4 был разработан Роном Ривестом в 1987 году. Этот шифр позволяет использовать ключи размером от 8 до 2048 бит (с шагом 8). В RC4 для зашифрования и расшифрования применяются одни и те же действия: генерируется гамма, которая накладывается на шифруемое сообщение путем сложения по модулю 2 (операция XOR).

RC4 применяется в таких продуктах, как Microsoft Office, Lotus Notes, Adobe Acrobat и др.

Алгоритм RC4 является собственностью компании RSA Data Security, Inc. Его описание никогда не было опубликовано и предоставлялось партнерам только после подписания соглашения о неразглашении. Однако в сентябре 1994 года в списке рассылки Cipherpunks (Шифропанки) кто-то анонимно опубликовал алгоритм шифрования, который на всех известных тестовых значениях совпадал с RC4. С тех пор сам алгоритм перестал быть секретом, но название RC4 остается торговой маркой. То есть, чтобы получить право заявлять, что в коммерческом программном продукте используется RC4, необходимо приобрести лицензию на этот алгоритм у RSA Data Security. А без лицензии можно утверждать лишь то, что "используется алгоритм, похожий на RC4 и совпадающий с ним на всем известном множестве тестов". Именно поэтому на языке ADA был реализован Alleged (предполагаемый) RC4.

Одно из достоинств RC4 (кроме обещаний компании RSA Data Security, Inc., что алгоритм устойчив к дифференциальному и линейному методам криптоанализа и, вероятно, не содержит коротких циклов) — его простота (листинг 7.1).

Листинг 7.1. Исходный текст алгоритма RC4 на языке г:

```
typedef unsigned char RC4_CELL;
typedef struct {        // структура для хранения текущего состояния ключа
  RC4_CELL state[256]; // таблица перестановок
  RC4_CELL x, y;
} RC4_KEY;

void swap_byte (RC4_CELL *a, RC4_CELL *b) { // обмен значений двух ячеек
  RC4_CELL t = *a; *a = *b; *b = t;
}

void RC4_setKey ( // загрузка ключа (инициализация таблицы перестановок)
  RC4_KEY *key,   // хранилище ключа
  int len,        // длина ключа
  RC4_CELL *data  // данные ключа


  RC4_CELL t, *s = key->State;
  int i, id;

  for (i = 0; i < 256; i++) s[i] = i;
  key->x = key->y = 0;

  for (id = i = 0; i < 256; i++) {
    id = (data[i % len] + s[i] + id) & Oxff;
    swap_byte (&s[i], &s[id]);



void RC4  (        // процедура шифрования
  RC4_KEY *key,   // хранилище ключа
  int len,        // длина шифруемых данных
  RC4_CELL *data // шифруемые данные
```

```
{
  RC4_CELL *s = key->state, x = key->x, y = key->y;

  for (; len > 0; len--, data++) {
    x = (x + 1) & 0xff;
    y = (s[x] + y) & 0xff;
    swap_byte (&s[x], &s[y]);
    *data ^= s[(s[x] + s[y]) & 0xff];
  }
  key->x = x; key->y = y;
}
```

Как видно, и в процедуре загрузки ключа, и в процедуре шифрования при вызове функции `swap_byte()` происходит обмен местами двух элементов таблицы перестановок.

Существует алгоритм обмена содержимого двух ячеек без использования вспомогательных переменных. Для того чтобы поменять местами `A` и `B`, необходимо выполнить три операции:

A = A xor B; B = B xor A; A = A xor B;

Именно этот прием и был использован в реализации RC4 на языке ADA. Однако автор кода не учел одну простую вещь: алгоритм обмена содержимого двух ячеек памяти без использования вспомогательных переменных работает только для разных переменных. Если значения, хранящиеся в A и в, совпадают, алгоритм работает правильно. Но если A и в — одна и та же переменная, ее значение будет обнулено при попытке обмена.

В RC4 индекс одного из переставляемых элементов таблицы последовательно (циклически) возрастает, а индекс другого элемента вычисляется по текущему состоянию ключа. И, разумеется, возникают ситуации, когда значения индексов совпадают. При этом в корректной реализации таблица перестановок просто остается без изменений, а при использовании обмена без вспомогательных переменных один из элементов таблицы будет обнулен.

Очень интересно посмотреть, как такая ошибка скажется на работе алгоритма шифрования в целом.

Во-первых, этот алгоритм может показаться полностью работоспособным, т. к. и зашифрование, и расшифрование будут идти совершенно одинаково. А значит, данные, зашифрованные этим алгоритмом, могут быть им же успешно расшифрованы.

Во-вторых, на коротких сообщениях алгоритм с ошибкой может работать в точности так же, как и правильно реализованный алгоритм. То есть ошибка может остаться незамеченной, если тестировать алгоритм шифрования только на коротких образцах.

И, в-третьих, при шифровании значительного объема данных без смены ключа все большее число элементов таблицы перестановок будет становиться нулевыми. А так как преобразование шифруемых данных осуществляется путем наложения выбранного элемента таблицы перестановок при помощи операции XOR:

```
*data ^= s[(s[x] + s[y]) & 0xff];
```

то шифртекст во многих местах будет совпадать с открытым текстом. То есть изрядная часть данных окажется вообще незашифрованной.

7.3. Многократное использование ключа потокового шифра

Использование правильно реализованного алгоритма шифрования еще не гарантирует, что данные будут действительно хорошо защищены. Так, весьма распространенная ошибка при использовании потоковых шифров — шифрование нескольких потоков на одном ключе.

Как было показано ранее, потоковые шифры вроде RC4 являются генераторами гаммы, которая накладывается на шифруемые данные путем сложения по модулю 2. Имея только зашифрованные данные, получить открытый текст без знания ключа практически невозможно. Однако, зная шифртекст и соответствующий ему открытый текст, не составляет труда вычислить фрагмент гаммы, соответствующей ключу. Это нормальная ситуация, т. к. хороший алгоритм шифрования по фрагменту гаммы не должен позволять определить ключ или другие фрагменты гаммы. Но если на том же самом ключе был зашифрован другой поток данных, то, зная шифртекст и фрагмент гаммы, элементарно вычислить открытый текст, соответствующий известному фрагменту гаммы.

Тот факт, что один и тот же ключ потокового алгоритма, работающего в режиме OFB, нельзя использовать дважды, можно отнести к азам криптографии, которые должен знать каждый, имеющий дело с информационной безопасностью. Однако даже разработчики, называющие себя профессионалами в области защиты программ от компьютерного пиратства, грешат невыполнением прописных истин.

Шифрование пользовательских данных в PACE InterLok

Система защиты программного обеспечения от компьютерного пиратства PACE InterLok разработана компанией PACE Anti-Piracy. На страницах интернет-сайта компании утверждается, что PACE Anti-Piracy имеет более 14 лет опыта в создании защит от пиратства (по состоянию на 2003 год).

Программы, защищенные с помощью InterLok, хранятся в зашифрованном виде и расшифровываются прямо в памяти в момент выполнения. Кроме того, специальный драйвер, устанавливаемый вместе с программой, не позволяет использовать отладчики во время выполнения программы. Также программисту доступны вызовы InterLok API, позволяющие, среди прочего, сохранять на диске секретные данные в зашифрованном виде, а потом их восстанавливать. Однако то, как используется шифрование, оставляет широкие возможности для атак.

Программа Adobe eBook Reader V2.2.203 защищена InterLok и использует InterLok API для того, чтобы сохранить секретную информацию, необходимую для извлечения персонального ключа пользователя. При сохранении блока данных через InterLok API все функции шифрования выполняет InterLok. Но, судя по всему, InterLok использует какой-то потоковый шифр, работающий в режиме OFB. Во всяком случае, определив один раз, какие именно данные сохраняются, и сложив их по модулю 2 с данными, записанными на диск, можно вычислить гамму. После этого, зная данные, хранящиеся на диске, легко наложить на них известную гамму и получить секретную информацию. Самое забавное то, что для получения доступа к секретным данным даже не требуется знать, каким алгоритмом они зашифрованы.

7.4. Ошибки в генераторах псевдослучайных чисел

Генераторы псевдослучайных чисел требуются практически в любом приложении, использующем криптографию. И очень часто разработчики не уделяют достаточно внимания характеристикам используемого генератора. Вот несколько иллюстраций.

Пожалуй, самый известный пример — ошибка в реализации протокола SSL (Secure Sockets Layer) в браузере Netscape.

Взлом 128-битового шифрования в Netscape

Компания Netscape разработала протокол SSL и реализовала его в своем браузере. Данные, передаваемые посредством SSL, зашифровывались алгоритмом RC4 со 128-битовым ключом. 17 сентября 1995 года Йен Голдберг (Ian Goldberg)

сообщил о том, что ему в сотрудничестве с Дэвидом Вагнером (David Wagner) удалось обнаружить уязвимость в процедуре выбора 128-битового ключа для алгоритма RC4. Недостаток процедуры заключался в том, что начальное состояние генератора псевдослучайных чисел основывалось на трех значениях: идентификаторе процесса, генерирующего ключ, идентификаторе его родительского процесса и текущем времени. Учитывая то, что значительную часть информации о номерах процессов и времени можно было предугадать, пространство возможных ключей сократилось с 2^{128} до 2^{20}, и на поиск ключа шифрования уходило всего 25 секунд.

Следующий пример снова связан с шифрованием в архивах формата ZIP. Атака на основе известного открытого текста применима только в том случае, если фрагмент открытого текста доступен (например, когда несколько файлов зашифровано с одним паролем и один из файлов есть в расшифрованном виде). Однако выяснилось, что в некоторых случаях удается получить достаточный для проведения атаки объем открытого текста из вспомогательных структур, создаваемых архиватором.

Генератор псевдослучайных чисел в InfoZIP

Согласно спецификации ZIP, после загрузки ключа необходимо зашифровать 12 байт до того, как начать шифровать данные файла. Последний из этих 12 байт является младшим байтом контрольной суммы файла, которая хранится в заголовке архива в незашифрованном виде. Это позволяет определять неправильный пароль с вероятностью $^{255}/_{256}$ после расшифровки всего 12 байт. Остальные байты обычно выбираются случайным образом.

Существует открытая реализация библиотеки для работы с архивами формата ZIP, называемая InfoZIP. В этой библиотеке из 12 дополнительных байт не один, а два последних байта содержат младшие байты контрольной суммы файла. Для генерации случайных байт используется алгоритм, идентичный `rand()` из стандартной библиотеки Microsoft Visual C++. Зная 4 байта выхода этого алгоритма, можно получить начальное состояние генератора и полностью предсказать его выход.

Псевдослучайные байты, полученные при помощи этого генератора, используются в InfoZIP таким образом, что для каждого файла в архиве генерируется 10 байт и один из них хранится в архиве в незашифрованном виде. Это позволяет при наличии в архиве пяти файлов, зашифрованных с одним паролем и подряд (без повторной инициализации генератора), найти начальное состояние генератора. А зная выход генератора и значения двух младших байт контрольной суммы для каждого из пяти файлов, можно определить ключ шифрования всех этих файлов менее чем за час.

Кстати, стоит отметить, что популярный архиватор WinZIP как раз основан на InfoZIP, а значит, созданные им архивы могут быть расшифрованы таким способом.

Последний пример связан с генерацией ключей RSA в программе, предназначенной для защиты других программ от несанкционированного тиражирования.

Ключи RSA-1024 в ASProtect

ASProtect представляет собой средство защиты программ от несанкционированного тиражирования, исследования и внесения изменений. Исполняемый файл при обработке его ASProtect зашифровывается и автоматически расшифровывается при загрузке в память. Но некоторые фрагменты, по желанию разработчика, могут оставаться зашифрованными до тех пор, пока не будет введен правильный регистрационный ключ.

Одна из возможностей, предоставляемых ASProtect, заключается в том, что разработчик генерирует пару ключей RSA-1024: открытый ключ хранится в программе, а секретный используется для генерации лицензий. Использование RSA-1024 обеспечивает невозможность генерации (и подделки) лицензий без знания секретного ключа.

По неподтвержденной информации, 1 января 2001 года Алексей Солодовников (автор ASProtect) получил от представителей группы DAMN по электронной почте генератор ключей к своей собственной программе. Примерно в это же время в Интернете были выложены генераторы лицензий еще к нескольким десяткам программ, защищенных ASProtect с использованием RSA-1024.

Взлом оказался возможен из-за того, что для генерации ключа RSA-1024 использовалась стандартная функция `rand()`, а начальное состояние генератора задавалось как:

```
(time(NULL) + GetCurrentThreadId()) ^ GetTickCount())
```

Похоже, в ASProtect использовалась некоторая криптографическая библиотека, в которой была функция `trueRandByte()`, которая просто возвращала `(unsigned char)rand()`.

В результате оказалось возможным подобрать ключ RSA-1024 ко многим программам. В настоящее время эта ошибка уже исправлена, и новые ключи не могут быть найдены подобным способом.

Далее приводятся алгоритмы (листинг 7.2), по которым функционируют некоторые широко используемые некриптографические генераторы псевдослучайных чисел. Коллекция собрана участниками форума www.reversing.net.

Листинг 7.2. Генераторы псевдослучайных чисел, входящие в стандартные библиотеки популярных языков программирования

```
static unsigned int seed;

// GCC/EMX
unsigned int emx_rand() {
  seed = seed * 69069 + 5;
  return (seed » 0x10) & 0x7FFF;


// Watcom C/C++
unsigned int wc_rand() {
  seed = seed * 0x41C64E6Du + 0x3039;
  return (seed » 0x10) & 0x7FFF;


// Borland C++ for OS/2
unsigned int bc2_rand() {
  seed = seed * 0x15A4E35u + 1;
  return (seed » 0x10) & 0x7FFF;
)
unsigned int bc2_lrand() {
  seed = seed * 0x15A4E35u + 1;
  return seed & 0x7FFFFFFF;


// Virtual Pascal == Delphi
unsigned int vp_random(unsigned int maxrand) {
  seed = seed * 0x08088405u + 1;
  return ((unsigned long long) seed * (unsigned long long)maxrand) » 0x20;
```

```
// Microsoft Visual C++
unsigned int rand() {
  seed = 0x343FD * seed + 0x269EC3;
  return (seed » 0x10) & 0x7FFF;
```

Как видно, все приведенные алгоритмы построены очень похоже, и ни один из них не стоит использовать для решения задач, требующих применения криптографии.

7.5. Блочный шифр в режиме простой замены

Если блочный шифр используется для шифрования больших объемов неслучайных данных, имеющих повторения, то, анализируя повторения в блоках шифртекста, можно получить некоторую информацию о содержимом файла.

Блочный алгоритм в режиме ECB в файлах SealedMedia

В зашифрованных PDF-файлах SealedMedia (с расширением spdf) легко обнаружить несколько одинаковых 8-байтных блоков, повторяющихся с периодом в 40 байт. Только на основании этой информации можно предположить, что используется блочный алгоритм шифрования с размером блока 8 байт (например DES), зашифрован PDF-файл, а повторяющиеся блоки относятся к таблице перекрестных ссылок. Каждая запись этой таблицы занимает 20 байт, и большинство элементов таблицы различаются в пяти-шести байтах.

Подтверждением этого предположения является, например, тот факт, что период повторений (40 байт) является наименьшим общим кратным размера блока шифра (8 байт) и размера записи таблицы (20 байт).

Разумеется, вся эта информация не позволяет быстро расшифровать документ, если используется стойкий алгоритм шифрования. Но если в будущем в алгоритме будут обнаружены серьезные недостатки, предположения о структуре зашифрованных данных могут дать некоторый объем открытого текста для атаки. А если бы при шифровании использовался другой режим, получить открытый текст было бы не так просто.

7.6. Использование обратимых хэш-функций

Для проверки целостности данных часто применяют цифровую подпись, например на основе алгоритма RSA. Подпись реализуется путем зашифрования значения хэш-функции от защищаемых данных на секретном ключе RSA. Для проверки целостности необходимо снова вычислить значение хэш-функции от тех же данных, а потом сравнить его со значением, расшифрованным открытым ключом RSA. Если при подписи используется плохая хэш-функция, то возможно подделать подписанные данные, не взламывая RSA.

Подпись файла данных в Hardwood Solitaire II

Программа Hardwood Solitaire II, разработанная компанией Silver Creek Entertainment, сочетает в себе несколько десятков карточных пасьянсов в великолепном графическом исполнении.

Для защиты основного файла данных от внесения изменений используется цифровая подпись на основе RSA по описанной выше схеме. Однако для вычисления хэша по непонятным причинам была выбрана функция `Adler32`, предназначенная для вычисления 32-битовой контрольной суммы.

Но при использовании `Adler32` добавлением максимум 260 байт к любым данным можно добиться того, чтобы результат вычисления контрольной суммы оказался равен любому заданному значению. Таким образом, можно в файл данных внести любые изменения, а затем дополнить этот файл таким образом, чтобы вычисление `Adler32` от него выдавало тот же результат, что и от оригинального файла. При этом цифровая подпись не будет разрушена.

7.7. Точное следование протоколам

При реализации криптографических протоколов некоторые моменты могут показаться программисту излишними, и он с легкой душой избавится от ненужного на его взгляд кода, тем самым еще и увеличивая быстродействие программы.

Например, при реализации шифрования по RSA кажется достаточным просто реализовать операцию модульного возведения в степень. Однако спецификация PKCS#1 (Public-Key Cryptography Standards) требует добавления к каждой порции шифруемых данных как минимум 8-ми случайных байт.

Дело в том, что при использовании алгоритмов с открытым ключом в распоряжении злоумышленника оказывается возможность расшифровывать все сообщения, зашифрованные на секретном ключе (что не дает ему значительного преимущества), а также самостоятельно зашифровывать любые сообщения на открытом ключе.

Допустим, злоумышленнику удалось перехватить зашифрованное на открытом ключе сообщение, в котором содержится только одно 32-битовое значение. Расшифровать это сообщение без знания секретного ключа злоумышленник не в силах, но он может перебирать все 2^{32} возможных значений, зашифровывая их на открытом ключе, пока не получит шифртекст, совпадающий с перехваченным. Если же при зашифровании были добавлены случайные байты, перебор 2^{32} вариантов уже не принесет желаемого результата.

На практике неполная реализация протокола — довольно распространенное явление. Разработчик может утешать себя тем, что все работает и так и никто не станет копаться в двоичном коде в поисках уязвимостей. Однако подобный подход может иметь тяжкие последствия при защите информации.

Цифровая подпись ElGamal в библиотеке FGInt

FGInt представляет собой библиотеку для работы с большими целыми числами и включает в числе прочих поддержку алгоритма цифровой подписи ElGamal. Но вычисление и проверка этой подписи в FGInt были реализованы с некоторыми отступлениями от спецификации.

Во-первых, при проверке целостности подписи должна выполняться проверка того, что значения двух составляющих подписи *r* и s не превышают значения использованного модуля *p*. На случай если эта проверка не выполняется, в книге "Handbook of Applied Cryptography" приводится алгоритм атаки, позволяющий при наличии одного подписанного сообщения вычислить цифровую подпись для любого другого сообщения.

Во-вторых, в подписи должно использоваться не само сообщение, а его хэш. Причина этого заключается в том, что без использования хэш-функции оказывается возможным подобрать сообщение, соответствующее заданному значению подписи, вычисленному определенным образом. Данная атака также описана в книге "Handbook of Applied Cryptography". А если подписывается значение хэша, то для отыскания подходящего сообщения придется обратить еще и хэш, что при использовании стойкой криптографической хэш-функции сделать почти невозможно.

Описанные выше недостатки в FGInt были обнаружены и успешно использованы представителями группы CORE, что позволило им выпустить генераторы

регистрационных кодов к нескольким версиям программы SmartWhois, защищенной с использованием цифровой подписи ElGamal с ключом длиной 960 бит, реализованной через библиотеку FGInt.

Подытоживая все написанное про алгоритмы и протоколы, можно сказать, что хотя криптографическая часть любой защиты, возможно, поддается формализации легче многих других элементов, это не делает ее гарантированно стойкой. Существует множество способов из надежных составляющих построить слабую систему. И только глубокие знания в криптографии, подкрепленные постоянной практикой, помогают снизить вероятность неудачи.



Глава 8

Рекомендации по выбору алгоритмов

На этапе составления спецификации разрабатываемого приложения, реализующего криптографические функции, необходимо принять решение, какие из множества возможных алгоритмов шифрования, вычисления хэш-функции или цифровой подписи использовать. И, разумеется, к выбору оптимального алгоритма надо подходить осознанно.

8.1. Конкурсы по выбору стандартов шифрования

Хорошим примером того, как стоит подходить к выбору алгоритма шифрования, может послужить описание процесса выбора алгоритма Advanced Encryption Standard (AES) — нового стандарта шифрования США, пришедшего на смену DES.

8.1.1. Стандарт шифрования США

В 1996 году национальный институт стандартов и технологий США (National Institute of Standards and Technology, NIST) начал работу над организацией конкурса по выбору лучшего алгоритма для нового стандарта.

2 января 1997 года NIST официально объявил о запуске программы по разработке нового федерального стандарта обработки информации (Federal Information Processing Standard, FIPS).

12 сентября 1997 года начался прием заявок на участие в конкурсе. К кандидатам предъявлялись следующие обязательные требования:

- алгоритм должен относиться к симметричной криптографии (с секретным ключом);
- алгоритм должен являться блочным шифром;
- алгоритм должен поддерживать следующие комбинации размеров ключа и блока шифрования: 128—128, 192—128 и 256—128.

15 июня 1998 прием заявок закончился, и к участию в конкурсе оказались допущены 15 алгоритмов, разработанных криптографами из 12 стран. В процессе сравнения конкурсантов оценивались следующие характеристики:

- криптостойкость — объем усилий, необходимых для успешного криптоанализа. Эта характеристика является наиболее важной для алгоритма шифрования и строится на основе следующих факторов:
  - реальная криптостойкость алгоритма по сравнению с конкурентами (при одинаковых размерах блока и ключа шифрования);
  - насколько выход алгоритма неотличим от случайной перестановки (пермутации) входного блока;
  - серьезность математического обоснования стойкости алгоритма;
  - другие факторы, проявившиеся во время открытого публичного изучения свойств алгоритма;
- стоимость реализации — затраты, с которыми придется столкнуться при использовании алгоритма. Стоимость определяется совокупностью следующих критериев:
  - лицензионные требования. Предполагается, что AES должен получить не меньшее распространение, чем имел DES, т. е. применяться массово. Следовательно, предпочтение отдается алгоритмам, которые или не покрываются патентами, или допускают неограниченное бесплатное использование в любой точке мира;
  - вычислительная эффективность (скорость алгоритма). Предполагается, что алгоритм должен быть достаточно быстрым как при программной, так и при аппаратной реализации;
  - требования к памяти. Оценивается объем необходимой постоянной и оперативной памяти при программной реализации в разных средах, а также число вентилей для аппаратной реализации;
- функциональные характеристики алгоритма, такие как:
  - гибкость. Предпочтение отдается алгоритму, допускающему более широкое применение, т. к. он способен решить больше задач, стоящих перед пользователями. Положительно характеризуют алгоритм следующие свойства:
    - возможность оперировать блоками и ключами шифрования, отличными по размеру от описанных в обязательных требованиях к алгоритму;
    - возможность надежной и эффективной реализации на большом количестве разных платформ: на восьмибитовых процессорах, в сетях асинхронной передачи данных (Asynchronous Transfer Mode, ATM), в голосовых и спутниковых коммуникациях, в телевидении высокой четкости (High Definition Television, HDTV) и т. д.;

- возможность использования алгоритма для построения эффективного потокового шифра, генератора аутентификационных кодов, хэш-функции, генератора псевдослучайных чисел и т. д.;

- удобство программной и аппаратной реализации. Алгоритм не должен быть спроектирован с ориентацией только на аппаратную или только на программную реализацию. Если алгоритм может быть реализован как встроенная программа (firmware), это является его достоинством;
- простота. Чрезмерное усложнение алгоритма затрудняет его анализ и реализацию, поэтому предпочтение отдается сравнительно простым алгоритмам.

Вычислительной эффективности и требованиям к памяти имеет смысл уделить чуть больше внимания.

Особенности программной и аппаратной реализации алгоритмов

Разумеется, и программно, и аппаратно можно реализовать практически любой алгоритм. Но при этом очень важно, как быстро будет работать программная реализация и насколько будет дешева и эффективна аппаратная.

Рассмотрим два алгоритма: RC4 и DES.

Программная реализация алгоритма шифрования RC4 очень проста *(см. разд. 7.2).* Внутреннее состояние шифра описывается 256-байтовой таблицей перестановок *state* и двумя регистрами *x* и *y.* Для шифрования одного байта необходимо произвести 3 операции чтения и 2 операции записи в таблицу перестановок, а также выполнить 3 операции сложения по модулю 256. И большая часть времени при шифровании тратится именно на обращение к памяти. Аппаратная реализация RC4 не будет давать значительных преимуществ перед программной, т. к. тормозящим фактором все равно останется обращение к памяти, а на современных универсальных (неспециализированных) компьютерах обмен с памятью сильно оптимизирован и производится на очень высокой частоте.

С алгоритмом шифрования DES все иначе. Исходный текст DES на языке С занимает десятки килобайт и весьма сложен для первичного восприятия. В процессе работы DES оперирует не байтами или словами, а отдельными битами. Но в процессорах архитектуры x86, к которым относятся Intel Pentium и AMD Athlon, практически нет готовых команд для работы с битами — каждую операцию DES приходится реализовывать посредством нескольких команд процессора. То же самое, наверное, справедливо для большинства современных универсальных процессоров. А при аппаратной реализации такие операции, как, например, перестановки битов, можно реализовать на схемном уровне,

> и они будут выполняться за один такт. И аппаратная реализация DES оказывается значительно быстрее, чем программная, даже несмотря на тактовые частоты современных универсальных процессоров, уже давно измеряемые в гигагерцах.
>
> Одна из важных областей применения нового стандарта шифрования — смарт-карты. Стоимость карты явным образом зависит от размеров доступной внутри карты оперативной и постоянной памяти. Самые дешевые смарт-карты стоят около 25 центов, но имеют всего 2 Кбайта постоянной памяти для хранения алгоритмов и констант и 64 байта оперативной памяти для хранения промежуточных значений и шифруемых данных. Разумеется, гораздо выгоднее иметь в качестве стандарта те криптографические алгоритмы, которые смогут работать в самых дешевых картах при минимальных ресурсах.

Вернемся к конкурсу AES. В 1999 году, после проведения второй конференции по выбору кандидата на AES, из 15 претендентов осталось только 5. Это были алгоритмы RC6, Rijndael, MARS, Serpent и Twofish. Безусловного лидера среди них не оказалось, но были явные кандидаты на выбывание. Так, например, Serpent оказался медленней всех конкурентов в программной реализации, а MARS — в аппаратной. Ни для одного из пяти финалистов не было предложено метода криптоанализа, позволяющего взломать алгоритм быстрее, чем полным перебором, но для RC6 был разработан способ взлома 15 из 20 используемых циклов шифрования.

В итоге, 2 октября 2000 года победителем конкурса был объявлен алгоритм шифрования Rijndael, разработанный бельгийцами Винсентом Райменом (Vincent Rijmen) и Йоханом Дайменом (Joan Daemen). Название алгоритма было образовано из начальных букв их фамилий.

26 ноября 2001 года NIST объявил о том, что в США Rijndael получил статус федерального стандарта обработки информации — нового стандарта шифрования данных.

8.1.2. Европейские криптографические схемы

AES — не единственный открытый конкурс по выбору стандарта шифрования. Аналогичный конкурс проводится также еврокомиссией и носит название NESSIE (New European Schemes for Signature, Integrity and Encryption, новые европейские схемы для цифровой подписи, контроля целостности и шифрования).

Размах у проекта NESSIE значительно шире, чем был у AES. В рамках NESSIE не просто ведется выбор симметричного блочного алгоритма,

но делается попытка подобрать коллекцию надежных и эффективных криптографических алгоритмов. Рассматриваются такие криптографические примитивы, как симметричные блочные и потоковые шифры, хэш-функции, алгоритмы контроля целостности сообщений, схемы цифровой подписи и шифрования с открытым ключом.

Проект NESSIE был начат в январе 2000 года. Предполагается, что после выработки рекомендаций по алгоритмам в отдельных категориях, NESSIE окажет существенное влияние на применение криптографии в Европе.

8.1.3. Стандарт ISO/IEC 18033

Параллельно с проектом NESSIE работы по стандартизации криптографических алгоритмов проводит и 27-ой подкомитет первого объединенного технического комитета международной электротехнической комиссии при международной организации по стандартизации (International Organization for Standardization / International Electrotechnical Commission, Joint Technical Committee 1 / Subcommittee 27, ISO/IEC JTC 1/SC 27). Указанный подкомитет самостоятельно не проводит сравнительного тестирования криптографических алгоритмов, но принимает решения, основываясь на информации, полученной в рамках таких открытых проектов, как NESSIE.

Выбранные алгоритмы должны стать частью стандарта ISO/IEC 18033. Есть основания полагать, что новый стандарт будет иметь много общего с рекомендациями NESSIE и существующими стандартами, такими как IEEE P1363.

8.1.4. Стандарт шифрования Японии

Агентство развития информационных технологий Японии (Information technology Promotion Agency, IPA), субсидируемое японским министерством международной торговли и индустрии, проводит оценку различных криптографических технологий в рамках проекта CRYPTREC. Целью проекта является формирование набора криптографических алгоритмов, которые будут использоваться в рамках программы электронного правительства Японии, инфраструктура которого должна быть сформирована в 2003 году.

CRYPTREC оценивает асимметричные криптографические технологии, пригодные для шифрования, аутентификации, цифровой подписи и распределения ключей, а также симметричные потоковые и блочные шифры, генераторы псевдослучайных чисел и хэш-функции.

8.2. Практические рекомендации известных специалистов

Когда для решения конкретной задачи требуется выбрать алгоритм, устраивать открытый конкурс — не самый быстрый путь. В большинстве случаев достаточно выбрать его из уже существующих алгоритмов.

Авторы книги "Криптография. Официальное руководство RSA Security" ("RSA Security's Official Guide to Cryptography") Стив Бернет (Steve Burnett) и Стивен Пэйн (Stephen Paine) для шифрования соединений между компьютерами рекомендуют использовать потоковый алгоритм RC4., т. к. он обеспечивает достаточно высокую скорость. Для таких приложений, как базы данных, электронная почта и защищенное (зашифрованное) хранение файлов, рекомендуется использовать блочные алгоритмы шифрования, в частности AES (Rijndael).

Брюс Шнайер (Bruce Schneier) вместе с Нилсом Фергюсоном (Niels Ferguson) в своей книге "Практическая криптография" ("Practical Cryptography") считают, что с точки зрения безопасности карьеры лица, принимающего решение, Rijndael — действительно самый приемлемый блочный алгоритм шифрования. Он принят в качестве стандарта в США, и если в AES в будущем будут обнаружены серьезные недостатки, наказывать за это, скорее всего, никого не станут. Кроме того, AES уже достаточно широко распространен и поддерживается большим числом библиотек. Так что AES — почти беспроигрышный выбор.

Если очень важно обеспечить максимальную стойкость шифрования, а скорость играет малозначительную роль, рекомендуется обратить внимание на другого финалиста конкурса AES — алгоритм Serpent, который большинство серьезных криптологов назвали самым надежным (или самым консервативным) из всех представленных на конкурс.

В качестве хэш-функции Шнайер и Фергюсон рекомендуют выбирать модификацию одного из алгоритмов семейства SHA (Secure Hash Algorithm). В настоящий момент существуют спецификации алгоритмов SHA-1, SHA-256, SHA-384 и SHA-512, вычисляющих хэш размером 160, 256, 384 и 512 бит соответственно: Модификация заключается в том, что хэш-функция от сообщения *m* вычисляется как SHA-x (SHA-x (/и)), что позволяет исключить атаку расширения длины сообщения (Length Extension), которой подвержены все хэш-функции семейства SHA и MD (MD2, MD4 и MD5).

Для асимметричного шифрования многие рекомендуют применять алгоритм RSA как, безусловно, самый распространенный асимметричный алгоритм.

RSA часто используется и для цифровой подписи, но кроме него существует множество других алгоритмов и схем, из которых наиболее широко распро-

странен стандартизованный в США алгоритм DSA (Digital Signature Algorithm). Чуть меньшую популярность имеет реализация DSA на эллиптических кривых, носящая название ECDSA (Elliptic Curves DSA).

8.3. Патенты на алгоритмы и протоколы

В России не действуют патенты на криптографические алгоритмы и протоколы. Но во многих странах такие патенты допускаются законодательством.

Если разрабатываемая программа будет использоваться не только на территории России, при выборе используемых в ней криптографических алгоритмов стоит оценить их применимость с точки зрения патентной чистоты.

Доступность открытого описания алгоритма еще не означает, что никто не станет предъявлять претензий, если этот алгоритм будет использоваться в коммерческом приложении. Как уже говорилось ранее, только экспертная оценка, выполненная силами криптографического сообщества, может дать некоторую уверенность в том, что алгоритм не содержит серьезных недостатков. А для того чтобы криптографы смогли изучать алгоритм, его необходимо открыто опубликовать.

Более того, даже если в Интернете присутствует бесплатная криптографическая библиотека, содержащая реализацию определенного алгоритма в исходных кодах, это еще не означает полной свободы в использовании алгоритма.

Патент на алгоритм RSA

Алгоритм RSA получил свое название по первым буквам фамилий авторов. Рональд Ривест (Ronald Rivest), Ади Шамир (Adi Shamir) и Леонард Аделман (Leonard Adelman) впервые опубликовали описание алгоритма в апреле 1977 года. Алгоритм RSA составляет существенную часть патента США № 4405829, выданного Ривесту, Шамиру и Аделману сроком до 20 сентября 2000 года. Но уже через 9 дней после получения патента эксклюзивная лицензия была предоставлена компании RSA Data Security, Inc., которая и выступала много лет как владелец прав на одноименный криптографический алгоритм.

Все желающие использовать алгоритм RSA в коммерческих приложениях должны были приобрести у RSA Data Security лицензию на криптографическую библиотеку BSAFE. Кроме BSAFE в RSA Data Security была разработана и бесплатная библиотека RSAREF, предназначенная для некоммерческого использования. Другим производителям не разрешалось распространять на территории США свои библиотеки, поддерживающие алгоритм RSA.

Однако с практической точки зрения к патентам можно относиться очень по-разному. Так, например, Шнайер и Фергюсон рекомендуют в своей книге не читать патенты и аргументируют это следующим образом.

Если вы не читали патент, а значит, не знали его содержимого, то в случае установления факта нарушения вам предстоит компенсировать нанесенный ущерб владельцу патента. Если же будет доказано, что вы предварительно ознакомились с патентом, то нарушение становится преднамеренным и может караться компенсацией в трехкратном размере. Таким образом, чтение патентов приводит к повышению степени ответственности в 3 раза.

Если вы после прочтения патента уверены, что ваши действия не нарушают патент, это еще не значит, что судья, рассматривающий иск о нарушении, придет к такому же заключению. Даже эксперт в некоторой технической области, к которой относится патент, не в состоянии судить, что покрывается патентом, а что нет. Это может сделать только патентный юрист, который берет за такую работу вознаграждение. Таким образом, чтобы не платить тройную компенсацию, придется платить юристу. Но патентов, которые могут случайно быть нарушены, очень много, и оплачивать услуги юриста для анализа каждого из них кажется не самым разумным решением. В результате получается, что для минимизации расходов на решение проблем с патентами самое разумное — это не читать патенты вообще, как ни парадоксально это звучит.



Часть III

КАК НЕ НАДО ЗАЩИЩАТЬ ПРОГРАММЫ

Эта часть книги посвящена вопросам защиты коммерческого программного обеспечения от нелегального тиражирования. В ней подробно рассматриваются наиболее популярные и эффективные решения, а также проблемы, связанные с их реализацией.



Глава 9

Актуальные задачи защиты программ

Глобальная цель защиты программных продуктов, по большому счету, всегда одна — повысить прибыль, получаемую от продажи программного обеспечения. Чаще всего это выражается в ограничении возможностей по распространению программы.

Однако в попытках достижения этой цели почти всегда возникает целый ряд подзадач, требующих решения.

9.1. Модели распространения программного обеспечения

Существует довольно много различных моделей распространения программного обеспечения. Рассмотрим характерные особенности некоторых их них.

9.1.1. Бесплатные программы (Freeware)

Эта модель распространения программного обеспечения подразумевает отсутствие оплаты за использование программ. Очень часто по такому принципу распространяются небольшие утилиты, которые, по мнению авторов, могут оказаться полезными широкому кругу пользователей, но не будут иметь спроса, если назначить плату за их использование.

Разумеется, существуют программисты, создающие бесплатные приложения из любви к искусству или нелюбви к излишней коммерциализации современных информационных технологий. Часто несколько добровольцев организуют команду, разрабатывающую и поддерживающую достаточно сложную программную систему.

Многие бесплатные программы распространяются в исходных текстах. Но надо отдавать себе отчет в том, что программы в исходных текстах и бесплатные программы — это совсем не одно и то же. Коммерческие продукты тоже могут поставляться в исходных текстах.

Довольно часто встречается ситуация, когда программа или библиотека бесплатна для личного использования, но требует лицензии (иногда весьма не дешевой) для коммерческого применения.

Не редки случаи, когда бесплатное программное обеспечение разрабатывается крупными коммерческими компаниями для упрочнения положения на рынке. Так, например, документы в формате PDF вряд ли имели бы сегодня такую популярность, если бы никогда не существовала бесплатная программа для их просмотра — Adobe Acrobat Reader, дополняющая линейку платных продуктов для создания PDF-документов (Acrobat Exchange, Distiller, Business Tools, Approval и т. д.). Аналогично, для документов, созданных в коммерческой программе Microsoft Word, существовала бесплатная программа просмотра Microsoft Word Viewer, также разработанная корпорацией Microsoft.

Иногда автор бесплатного проекта продает кому-нибудь все права на свое детище, а новый владелец начинает распространять ту же самую программу за деньги, иногда нанимая бывшего автора для продолжения разработки. Не удивительно, что такая судьба, в подавляющем большинстве случаев, постигает именно удачные и полезные бесплатные программы.

9.1.2. Почти бесплатные программы

Иногда авторы программ по каким-то соображениям не хотят распространять их как коммерческие продукты, но и не возражают против получения какой-нибудь отдачи, не считая морального удовлетворения. Чаще всего выбирается один из следующих методов распространения".

- *Cardware* — каждый пользователь программы, желающий зарегистрироваться, должен послать автору программы почтовую открытку с видом местности, где он проживает;
- *Mailware* — более современный вариант Cardware, подразумевающий отсылку автору электронного письма. Как правило, в ответ автор присылает регистрационный код, дающий возможность работать с программой;
- *Donationware* — это когда автор не требует никакой оплаты, но предлагает всем, кому понравилась программа, пожертвовать произвольную сумму, чтобы поддержать разработку;
- *Giftware* — почти то же самое, что и Donationware, но автор готов принимать не только денежные пожертвования, но и другие подарки;
- *Beerware* — благодарность за программу принимается в виде пива;
- *Vegeware* — автор собирает с пользователей плату за программу в форме рецептов вегетарианских блюд;

- *Memorialware* — человек по имени Гари Крэмблитт (Gary Cramblitt) посвятил свою программу памяти отца и распространяет ее как Memorialware. Для пользователей программа бесплатна, но всем желающим предлагается помочь мемориальному фонду Крэмблитта-отца.

9.1.3. Программы, показывающие рекламу (Adware)

В последние годы XX века, когда бурный рост интернет-технологий еще не успел перейти в глубокий кризис, была популярна модель распространения программного обеспечения, демонстрирующего рекламу. "Ad" является сокращением от английского слова Advertisement — реклама.

Основная идея заключалась в том, что разработчик получал плату за использование программы не от конечного потребителя, которому программа доставалась бесплатно, а от рекламодателей. А пользователь был вынужден смотреть на доставляемые программой через Интернет рекламные картинки. Разумеется, этот подход годился преимущественно для программ, работа которых прямо связана с доступом в Интернет.

Однако со временем эффективность рекламы такого рода значительно снизилась, и найти желающих платить за нее настоящими деньгами стало довольно трудно. Но продолжают существовать спонсируемые программные продукты (как правило, информационной направленности), разработка которых ведется на деньги рекламодателей в обмен на размещение их информации в программе.

9.1.4. Коммерческие программы (Commercial)

Коммерческое программное обеспечение, очевидно, создается с целью извлечения прибыли и распространяется за материальное вознаграждение. Наверное, коммерческие программы больше всего похожи на обычные товары, которые люди привыкли покупать в магазинах.

Прежде всего, для программного обеспечения, распространяемого как чисто коммерческое, применяется принцип "деньги — вперед", т. е. пользователь получает программу только после полного внесения оплаты.

Очень многие программы, распространяемые таким способом, являются "коробочными" — при покупке пользователь получает коробку, в которой уложены носители информации (например DVD или компакт-диски), документация, регистрационная карточка и еще все что угодно, на усмотрение продавца.

Разумеется, автор (или правообладатель) кровно заинтересован в том, чтобы собрать плату с каждого пользователя программы. Для достижения этого необходимо применять технологические методы, ограничивающие распространение нелегальных (нелицензированных) копий программы. К популярным технологическим методам относятся различные аппаратные защиты, системы регистрации и активации, проверка лицензии через Интернет при каждом запуске программы и т. д.

Однако чисто коммерческое программное обеспечение обладает одной очень важной особенностью. Конечный потребитель сможет получить представление о том, что он покупает, только после совершения покупки, и, следовательно, достаточно высока вероятность того, что он будет разочарован и захочет избавиться от программы и вернуть назад заплаченные деньги. Для того чтобы не отпугнуть покупателей, продавцы часто вынуждены обещать возврат денег в течение, например, двух недель с момента покупки, если программа не понравится.

9.1.5. Почти работоспособные программы

Разработчики коммерческих программ в рекламных целях выпускают ограниченные ознакомительные версии своих продуктов. Такие версии обычно не позволяют плодотворно работать, но создают правдивое впечатление о функциональности программы. Можно выделить несколько основных типов ограниченных программных продуктов:

- *Demoware* — это когда в программе присутствуют функциональные ограничения. Например, можно обрабатывать файлы не более определенного размера, нельзя выполнять сохранение и т. д. Такие программы иногда называют *Crippleware* — "урезанное" программное обеспечение;
- *Trialware* — подразумевается наличие ограничений по времени использования. Ограничения могут выражаться в виде длительности периода времени, на протяжении которого можно пользоваться программой (например 30 дней с момента инсталляции) или в виде фиксированной даты истечения тестового периода. Может ограничиваться число запусков программы или число процессов обработки;
- *Nagware* — пользователь регулярно извещается о том, что данная версия программы не является полноценной коммерческой версией. Такое извещение может выглядеть как диалоговое окно, появляющееся при запуске программы и с некоторой периодичностью во время работы, дополнительные надписи, выводимые на принтер или экран, и т. д.

Разумеется, возможны различные комбинации описанных ограничений. И далеко не каждый коммерческий продукт имеет ознакомительную версию — такой подход скорее исключение.

9.1.6. Условно бесплатные продукты (Shareware)

Наличие возможности оценить программу до покупки ("try before you buy") является отличительной чертой условно бесплатных продуктов. *Share* переводится с английского языка как разделять, совместно использовать. Подразумевается, что незарегистрированную версию условно бесплатной программы можно свободно распространять в неизмененной форме.

Условно бесплатное программное обеспечение так же, как и коммерческое, разрабатывается с целью извлечения прибыли. Но потенциальному пользователю до того, как он заплатит деньги, обязательно предоставляется возможность попробовать программный продукт в действии в течение некоторого периода времени.

По истечении тестового периода потенциальный покупатель должен принять решение о приобретении программы. Если решено отказаться от покупки, то надо прекратить использовать программу и удалить ее с компьютера. В противном случае, необходимо оплатить лицензию на программу, после чего продавец предоставит возможность получения полнофункциональной версии программы.

Обычно условно бесплатные программы доставляются через Интернет и имеют небольшой размер (единицы мегабайтов). Также очень часто для превращения ограниченной версии в полнофункциональную не требуются никакие дополнительные файлы — достаточно ввести правильный регистрационный код, полученный от продавца.

На период бесплатного использования накладываются ограничения, схожие с ограничениями на оценочные версии коммерческих программных продуктов. Точные ограничения оценочного периода, как правило, оговариваются в лицензионном соглашении на использование каждого конкретного продукта.

Условно бесплатные продукты очень популярны. Многие крупные разработчики берут на вооружение концепцию "try before you buy", чтобы заинтересовать покупателей. По сути, ограниченные ознакомительные версии коммерческих продуктов являются всего лишь одной из модификаций идеи Shareware. Даже корпорация Microsoft бесплатно распространяет 120-дневные версии Windows 2003 Server и Visual Studio .NET. Правда, небольшое отличие заключается в том, что для превращения 120-дневной версии в полноценную придется получить диск с новой версией и выполнить процедуру обновления, а для "классических" условно бесплатных программ переход к полной версии происходит сразу же после ввода правильного регистрационного кода.

9.2. От чего защищают программы

Коммерческие программы обычно защищают от несанкционированного тиражирования.

Наличие доступа только к носителю информации с дистрибутивом (набором инсталляционных файлов) программного продукта не должно давать возможности установить работоспособную копию программы. То есть данных дистрибутива, который можно скопировать или незаметно взять на несколько дней, не должно хватать для создания работоспособной копии программы. Подобные ограничения могут быть реализованы разными способами. Например, очень многие коммерческие программы при инсталляции требуют ввести серийный номер, напечатанный на коробке или указанный в одном из прилагаемых к программному продукту документов (у Microsoft — в сертификате аутентичности).

Также часто возникает потребность ограничить число пользователей, одновременно работающих с программой. То есть человек, который приобрел лицензию на одно рабочее место, не должен иметь возможности создать 2 рабочих места, функционирующих одновременно. Это достигается за счет использования аппаратных ключей, менеджеров лицензий и процедуры активации.

Для некоторых программных продуктов (в частности игр) часто используется привязка к носителю информации, например компакт-диску. То есть для запуска игры требуется наличие в приводе оригинального компакт-диска, который защищен от копирования стандартными средствами.

Для оценочных версий, ограниченных по времени или числу запусков, необходимо правильно реализовать хранение счетчиков, чтобы злоумышленник не смог заставить работать программу, просто переведя часы или удалив файл, в который записывается количество запусков программы или число обработанных файлов.

Условно бесплатные продукты, в отличие от ограниченных по функциональности оценочных версий коммерческих программ, после ввода регистрационного кода должны предоставлять доступ ко всем функциям, предусмотренным в полной версии программы. То есть в бесплатно распространяемой версии программы должны быть реализованы все функции полной версии. Следовательно, желательно так организовать защиту, чтобы злоумышленник не смог добраться до функций, присущих только полной версии, пока в его распоряжении не будет правильного регистрационного кода.

Процедуры проверки правильности серийных номеров, а также регистрационных кодов и кодов активации должны строиться таким образом, чтобы злоумышленник не мог самостоятельно генерировать правильные коды и, в то же время, длина кодовой строки не была очень большой.

Также может возникнуть потребность защищать любые исполняемые файлы от внесения изменений, дизассемблирования, исследования под отладчиком и т. д.

9.3. Основные форматы исполняемых файлов

За время существования персональных компьютеров на процессорах семейства x86 (начиная от IBM PC XT на процессоре Intel 8086) успело смениться несколько форматов двоичных файлов, предназначенных для хранения откомпилированного кода программы.

В операционной системе DOS (Disk Operating System) поддерживалось два основных формата исполняемых файлов: COM и EXE. COM-файлы загружались в оперативную память без каких-либо дополнительных настроек, и их размер не должен был превышать 64 Кбайта. EXE-файлы не имели таких жестких ограничений на размер и состояли из заголовка, включающего всю необходимую информацию для правильной загрузки программы в память, и собственно кода программы. Заголовок DOS-овских EXE-файлов начинался с символов 'MZ' или 'ZM', и до сих пор его так и называют — MZ Header (MZ-заголовок). Буквы MZ являются инициалами Марка Збыковски (Mark Zbikowski), являвшегося разработчиком данного формата. Сейчас все исполняемые файлы содержат MZ-заголовок, за которым может следовать информация о другом формате.

С появлением 16-битовой версии Windows возникла потребность в расширенном формате исполняемых файлов. В Windows была реализована поддержка динамически подсоединяемых библиотек (dynamic-link library. DLL), поэтому новый формат должен был обеспечить возможность хранения, в частности, таблиц экспортируемых (находящихся в DLL и доступных другим модулям) и импортируемых (находящихся во внешних библиотеках) функций. Кроме этого в Windows широко используются ресурсы — двоичные данные, содержащие иконки, курсоры, описания диалогов и т. д., которые желательно хранить внутри исполняемых файлов. Все актуальные на тот момент требования были учтены при разработке формата, получившего название New Executable (NE, новый исполняемый файл). Заголовок такого файла начинается с символов 'NE'.

Для хранения драйверов виртуальных устройств Windows (Virtual device driver, VxD) применялся формат Linear Executable (LE, линейный исполняемый файл). Его модификация под названием Linear executable (LX) использовалась для хранения исполняемых файлов, используемых в операционной системе OS/2 начиная с версии 2.0.

В связи с появлением 32-битовой операционной системы Windows NT в Microsoft был разработан формат Portable Executable (РЕ, переносимый исполняемый файл). Точнее, был доработан до своих нужд взятый за прототип формат объектных файлов COFF (Common Object File Format), используемый в Unix. Слово "переносимый" в названии, скорее всего, было призвано отражать тот факт, что один и тот же формат файла использовался как во всех 32-битовых операционных системах Microsoft на платформе х86, так и в Windows NT на других платформах (MIPS, Alpha и Power PC). Этот формат является основным для всех современных версий Windows, и именно ему в книге будет уделено наибольшее внимание.

9.4. Особенности внутреннего устройства исполняемых файлов в 32-битовых версиях Windows

Не вдаваясь слишком глубоко в детали формата Portable Executable, отметим только те элементы, которые наиболее часто подвергаются воздействию в процессе защиты.

Заголовок РЕ-файла содержит очень большое число различных полей и таблиц. Одно из полей определяет так называемую *точку входа* (Entry Point) — то место в программе, куда будет передано управление после загрузки программы в память. В DLL управление передается на точку входа не только при загрузке библиотеки, но и при ее выгрузке из памяти, а также при создании и уничтожении потоков выполнения внутри программы.

Обычно исполняемый файл состоит из нескольких *секций* (точное количество секций указано в РЕ-заголовке). Редактор связей (Linker), как правило, объединяет в одну секцию однотипную информацию.

Типичный исполняемый файл содержит секции *кода,* статических и динамических *данных* и секцию *ресурсов*. Каждая секция описывается именем, размером и положением в файле и в памяти, а также набором атрибутов (двоичных флагов): код или данные в секции, данные инициализированы или нет, разрешено ли исполнение, чтение или запись и т. д.

В секции кода помещаются собственно команды процессора, исполняемые в ходе работы программы. Эта секция имеет атрибуты, разрешающие исполнение кода и запрещающие запись.

Статические данные не изменяются после загрузки программы в память. поэтому для секции статических данных обычно не устанавливается атрибут. разрешающий запись. Попытка записи в эту секцию в момент выполнения

программы приведет к выработке исключительной ситуации (exception). Атрибуты секции динамических данных, напротив, разрешают запись в нее.

Секция ресурсов также не должна изменяться в процессе выполнения программы, и поэтому она не имеет атрибута, разрешающего запись.

Разумеется, выше была приведена лишь одна из возможных схем разбиения содержимого исполняемого файла по секциям, и каждое средство разработки, создающее PE-файлы, вправе самостоятельно решать, какая информация в какой секции окажется. Более того, вся программа может размещаться в одной секции и быть при этом вполне работоспособной.

Кроме описания секций в заголовке PE-файла присутствует специальный каталог (PE Directory), описывающий размер и положение служебных структур, необходимых для правильной загрузки программы в память. Эти структуры определяют, в каком месте программы хранятся ресурсы, как искать адреса экспортируемых функций, как настраивать ссылки на *импортируемые функции* и т. д.

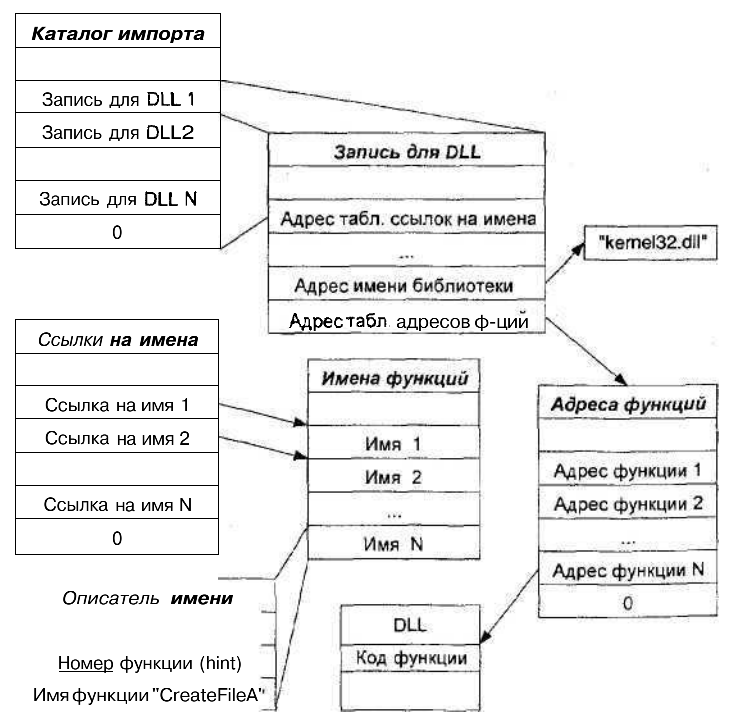


Рис. 9.1. Таблицы импорта

Импортируемые функции, т. е. функции, код которых находится в других исполняемых модулях, но используется во время работы программы, описываются посредством таблиц импорта. Это четыре связанных между собой таблицы: каталог импорта (Import Directory Table), таблица ссылок на имена функций (Lookup Table), таблица имен функций (Hint-Name Table) и таблица адресов импортированных функций (Import Address Table, IAT).

Таблицы импорта предназначены для того, чтобы правильно заполнить все значения в таблице адресов импортированных функций. Каждое из этих значений представляет собой адрес определенной функции во внешней библиотеке после ее загрузки в адресное пространство процесса (рис. 9.1).

PE-файлы содержат еще много разных таблиц и атрибутов. Более подробная информация может быть найдена в технической документации на этот формат.

Глава 10



Регистрационные коды для программ

Процедура регистрации приобретаемой продукции существует в мире довольно давно. После совершения покупки потребитель заносит некоторые сведения о себе в регистрационную карточку и отправляет ее производителю. Таким образом, потребитель становится зарегистрированным пользователем и получает все причитающиеся ему привилегии, например техническую поддержку и гарантийное обслуживание, а производитель пополняет статистическую информацию о своих клиентах. Многие "коробочные" программные продукты также содержат регистрационные карточки, а в последнее время даже позволяют отсылать регистрационную информацию через Интернет.

Так как имя пользователя не является уникальным, каждый экземпляр продаваемой продукции целесообразно связывать с некоторым неповторяющимся значением, называемым серийным номером. Этот номер указывается пользователем при заполнении регистрационной карточки и в дальнейшем используется при общении с производителем. А в приложении к программным продуктам серийный номер вполне может выполнять и вспомогательную функцию — ограничивать нелегальное копирование. Если программа при установке требует ввести правильный серийный номер, украв (скопировав) носитель с дистрибутивом программы, который одинаковый у всех пользователей, получить рабочую копию программы не удастся. А распространение серийного номера позволяет найти и наказать ассоциированного с этим номером пользователя.

В некоторых случаях после установки программы (неважно, с вводом серийного номера или без) для получения доступа ко всем функциям программы пользователю необходимо выполнить еще одну процедуру — регистрацию или, как это теперь называет Microsoft, активацию. Такое поведение характерно для большинства Shareware-продуктов, а также для программ, разработчики которых считают, что пользователь не имеет права работать, пока не сообщит о себе все необходимые сведения, даже если он уже приобрел лицензию.

Иногда серийный номер передается пользователю не в явном виде, а как часть кода активации/регистрации.

10.1. Требования и классификация

Программа должна содержать некоторый механизм, позволяющий проверить, является ли введенный пользователем серийный номер (или регистрационный/активационный код) правильным, а возможность вычислять правильные коды должна всегда оставаться только в руках разработчика.

Для того чтобы противник, исправив несколько байт, не смог заставить программу работать так, как будто она была корректно зарегистрирована активирована, необходимо те фрагменты кода или данных, доступ к которым разрешен только легальным пользователям, зашифровать стойким алгоритмом, а ключ шифрования вычислять, используя регистрационный код. Тогда без знания регистрационного кода получить полноценную версию программы не удастся. Подобную функциональность обеспечивают, например, программы ASProtect (ASPack Software) и EXECryptor (SoftComplete Development).

Определим несколько критериев, по которым можно сравнивать свойства разных методов генерации и проверки кодов:

- возможность связать код с именем пользователя или характеристиками компьютера;
- невозможность вычислить какой-нибудь правильный код, имея в распоряжении только алгоритм проверки;
- невозможность вычислить код для определенного пользователя, зная алгоритм проверки и правильный код другого пользователя;
- невозможность расшифровки программы (получения ключа шифрования) при наличии заблокированного (занесенного в черный список) регистрационного кода;
- длина ключевой строки (удобство пользователя).

10.2. Методы проверки регистрационных кодов

Все методы проверки правильности кодов можно, условно, разделить на три категории:

- алгоритмические, основанные на принципе "черного ящика";
- алгоритмические, основанные на математически сложной задаче;
- табличные.

10.2.1. "Черный ящик"

Любые алгоритмические методы позволяют связать код с именем пользователя или информацией о его компьютере, тем самым усложнив получение нескольких копий программы, выглядящих как легальные.

При использовании "черного ящика" разработчик старается запутать алгоритм проверки, чтобы его было труднее понять и обратить. Такой подход, наверное, используется чаще всех остальных вместе взятых. Причем его применяют не только неопытные разработчики. Так, например, именно в надежде на то, что противник не сможет разобраться, что к чему, изначально строилась система лицензирования FLEXlm, разрабатываемая компанией Globetrotter, а теперь принадлежащая Macrovision. Но противник, зачастую, оказывается одареннее, упорнее и лучше подготовлен в своей профессиональной области, чем разработчик защиты. Возможно, именно поэтому на очень многие продукты, защищенные FLEXlm, в Интернете нетрудно найти поддельные лицензии. А начиная с версии 7.2, FLEXlm поддерживает лицензии на основе эллиптических кривых, поделка которых сводится к решению сложной математической задачи.

Если процедура проверки написана без грубых ошибок, то получить из нее правильный код невозможно, но, зная один правильный код и обратив процедуру проверки, взломщик может вычислить любые новые коды.

Правда, попытки запутать алгоритм проверки в последнее время получают научную поддержку. Так на семинаре по проблемам управления цифровыми правами DRM Workshop 2002 была представлена работа "A White-Box DES Implementation" ("Прозрачная реализация DES"), в которой предлагалась реализация DES, где ключ шифрования не фигурирует в явном виде, но является частью кода, обрабатывающего данные. То есть при смене ключа изменится код функции шифрования. Подобный подход, возможно, является перспективным, но в рамках того же DRM Workshop 2002 была представлена другая работа, в которой анализировалась White-Box DES-реализация и предлагался метод эффективной атаки, приводящей к извлечению ключа шифрования.

10.2.2. Сложная математическая задача

Алгоритмические методы проверки регистрационного кода, основанные на сложной математической задаче, не нуждаются в сокрытии деталей реализации. Их особенность в том, что для генерации кода и проверки его правильности используются два разных алгоритма и получение алгоритма генерации из алгоритма проверки является математической задачей, не

имеющей на настоящее время эффективного решения. Чаще всего для этих целей используются криптографические алгоритмы с открытым ключом.

Однако у асимметричной криптографии есть одна особенность: размер блока, над которым производятся операции, довольно большой. Так, для RSA-1024 размер блока (а значит, и минимальный размер регистрационного кода) составляет 128 байт. А чтобы двоичные данные можно было ввести с клавиатуры, их кодируют, например, алгоритмом MIME64, который увеличивает размер блока на треть. То есть длина кодовой строки составит как минимум 172 символа. Очевидно, что ввести без ошибок бессмысленную последовательность такой длины, состоящую из букв, цифр и знаков препинания, почти невозможно. Это делает данную схему неприемлемой для регистрации, например, по телефону или факсу.

Делаются попытки создать стойкую систему регистрации, использующую короткие коды и основанную на сложной математической задаче. Так, компания SoftComplete Development в своем продукте HardKey System версии 2.0 (апрель 2002) использовала алгоритм, для взлома которого надо было многократно вычислить дискретный логарифм, что является сложной задачей. Существует изящный способ вычисления дискретного логарифма, который требует времени и памяти пропорционально квадратному корню из модуля. С его помощью на однопроцессорной машине с тактовой частотой 2 ГГц версия HardKey, использующая 35-битовый модуль, могла быть взломана примерно за 10 минут, а 47-битовый модуль — за сутки. Но для взлома версии, использующей 62-битовой модуль, потребовалось бы 16 Гбайт оперативной памяти, что рядовой взломщик себе вряд ли сможет позволить. Тем не менее, к ReGet Deluxe версии 3.118 RC, использовавшей именно 62-битовый модуль, группой SSG был создан генератор кодов. Правда, по непроверенной информации, вычисление дискретного логарифма не проводилось — секретный ключ якобы был похищен с сервера авторизации. Но современные оценки сложности вычисления дискретного логарифма совпадают с оценками сложности для разложения числа на простые множители, выполняемого при взломе RSA. И, как известно, еще в 1999 году был факторизован 512-битовый ключ RSA, а значит, теоретически на современной технике можно вычислить и дискретный логарифм по 512-битовому модулю.

HardKey System начиная с версии 3.0 опирается на другую сложную математическую проблему — Hidden Field Equations (HFE, замаскированная система уравнений над полем), которая, на настоящий момент, очень слабо изучена, но вроде бы позволяет обеспечить достаточно высокий уровень стойкости при малом размере блока. Однако стоит заметить, что HFE является патентованной технологией во многих странах.

В любом случае, при правильной реализации алгоритмические методы, основанные на математически сложной задаче, не позволяют взломщику, знающему один правильный регистрационный код, генерировать новые коды. Но единственный способ заблокировать украденный код заключается в том, чтобы занести этот код в так называемый "черный список". И очевидно, что обход блокировки требует не решения математической задачи, а исправления логического условия. То есть при желании взломщик может получить полностью работоспособную версию расшифрованной программы, имея только заблокированный код.

10.2.3. Табличные методы

В табличных методах генерируется заданное количество регистрационных кодов (по числу возможных пользователей), и в программе хранятся таблицы, построенные на основе этих кодов. Разумеется, коды не могут зависеть от имени пользователя или характеристик системы, т. к. генерируются до того, как появляются первые зарегистрированные пользователи.



Рис. 10.1. Формирование записей в таблице ключей



Рис. 10.2. Проверка регистрационного кода

амый простой способ — хранить в программе результат вычисления крип- ографической хэш-функции от каждого кода. При этом легко проверить правильность ключа, вычислив его хэш, но, имея значение хэша, вычислить ключ практически невозможно.

Если часть регистрационного кода сделать статической (одинаковой для всех кодов), то ее можно использовать в качестве ключа для шифрования программы. Но при таком подходе заблокированный код легко может быть использован для расшифровки, если его хэш добавить в таблицу, хранимую внутри программы, или вообще отключить проверку правильности хэша.

Поэтому более правильно для каждой новой публичной версии продукта случайным образом выбирать ключ шифрования программы. А каждая

запись таблицы должна получаться путем зашифрования ключа программы на ключе, полученном из регистрационного кода. И, разумеется, каждая запись должна содержать некоторую контрольную информацию, позволяющую оценить правильность расшифровки.

На рис. 10.1 приведена блок-схема алгоритма, используемого для формирования записей, соответствующих отдельным регистрационным кодам. Рисунок 10.2 иллюстрирует обратный алгоритм — проверку правильности регистрационного кода и извлечение ключа шифрования программы.

При таком подходе каждый регистрационный код никак не зависит от всех остальных, но позволяет легко вычислить ключ шифрования программы. А для блокировки кода достаточно удалить соответствующую ему запись из таблицы.

Если выяснится, что количество пользователей может превысить количество сгенерированных регистрационных кодов, таблица может быть увеличена до любого желаемого размера. Правда, владельцы новых (добавленных) регистрационных кодов не будут признаваться старыми версиями программы как лицензированные пользователи.

10.3. Какой метод выбрать

У каждого из описанных ранее методов проверки правильности кодов есть свои достоинства и недостатки.

Так "черный ящик" сравнительно прост в реализации и позволяет использовать короткие коды, привязанные к имени пользователя. Но почти всегда при использовании этого подхода возможно создание генератора ключей.

Методы, основанные на стойкой криптографии, весьма сложны для самостоятельной реализации и очень часто требуют длинной кодовой строки. Кроме того, многие алгоритмы покрываются действующими патентами.

Только табличные методы дают возможность полностью блокировать скомпрометированные регистрационные коды, но не позволяют привязывать код к имени пользователя. Кроме того, если число пользователей слишком велико, таблицы могут занимать значительный объем.

В общем, при выборе оптимального метода генерации ключей в каждом конкретном случае стоит руководствоваться свойствами продукта, характеристиками потенциального рынка и т. д. Но одного "самого хорошего" метода, годящегося на все случаи жизни, не было, нет и никогда не будет.

ава 11



ивязка
осителям информации

тьно на поверхности лежит идея привязки программы к носителю ин- ации, на котором программа доставляется пользователю. Действитель- › прихода Интернета в жизнь рядового пользователя программы рас- занялись преимущественно на дискетах, затем — на компакт-дисках. ого чтобы ограничить возможности пользователя по тиражированию ьно приобретенной программы, хорошо бы добиться того, чтобы про- а запускалась и работала только тогда, когда пользователь вставит :овод оригинальный диск.

я близкая по сути идея подразумевает распространение информации резаписываемых носителях и наличие на каждом оригинальном носи- iекоторого счетчика установок. При инсталляции программы счетчик шается до нуля, и выполнение дополнительных инсталляций стано- невозможным. При деинсталляции счетчик снова увеличивается, что іяет снова установить программу на другую машину.

:х случаях требуется некоторый способ создания и контроля уникаль- носителя.

. Ключевые дискеты

емена DOS существовало несколько способов создания некопируемых вых дискет.

из способов — использование дорожек с нестандартным размером ра. По историческим причинам почти все перезаписываемые носители рмации на компьютерах, которые назывались IBM-совместимыми, размер сектора, равный 512 байтам. В базовую подсистему ввода- а (Basic Input-Output System, BIOS) были встроены возможности рабо- :екторами других размеров, но для DOS в пределах одного диска все за должны были иметь один и тот же размер.

Защита заключалась в том, что одна из дорожек, не содержащая полезных данных, форматировалась с нестандартным размером сектора, например 128 или 256 байт, и на нее записывалась ключевая информация. Для DOS, уверенной в неизменности размера сектора не протяжении всего диска, эта дорожка выглядела поврежденной, но программа через BIOS могла читать и записывать на нее необходимые данные. Очевидно, что такой подход позволял довольно легко повторять манипуляции на уровне BIOS и тиражировать ключевые дискеты.

Кстати, если защита использовала счетчик инсталляций, очень часто ее можно было обмануть с помощью простого трюка. Достаточно было перехватить сервисное прерывание, отвечающее за обращение к дискам, и на все запросы записи и форматирования дорожек дискеты возвращать статус "успешно", не выполняя никаких действий. Затем дискета защищалась от записи, и запускался процесс установки. И редко какая защита после получения информации об успешной записи уменьшенного значения счетчика проверяла, действительно ли новое значение меньше предыдущего.

Схему, позволяющую восстанавливать значение счетчика при деинсталляции, можно обойти и другим способом. Сразу после установки программы необходимо сделать полную копию содержимого жесткого диска, деинсталлировать программу и восстановить образ диска с копии. При этом на жестком диске должна оказаться работоспособная версия установленной программы, а счетчик инсталляций при этом останется в исходном состоянии.

Другой способ создания некопируемых дискет заключался в нанесении физических повреждений на магнитный слой. Крупные производители использовали для этого лазеры. А разработчики-одиночки не менее успешно достигали того же результата при помощи любого острого предмета, например булавки.

Защита пыталась читать или писать заранее известную область диска, и если запрос проходил без ошибки, то дискета не признавалась подлинной.

При наличии некоторой практики можно довольно точно воссоздавать повреждения на другой дискете, но гораздо проще было эмулировать поврежденные сектора, т. к. их список было довольно легко получить, если имелся доступ к оригинальной дискете.

Несмотря на то, что дискеты практически вышли из обихода с появлением перезаписываемых компакт-дисков, последний раз защиту, основанную на использовании испорченных секторов на дискете, мне довелось увидеть в январе 2001 года. Так был защищен один из пакетов программ, предназначенных для восстановления забытых паролей.

Самый продвинутый способ создания ключевых дискет реализовывался на уровне портов ввода-вывода контроллера гибких дисков — самом низком

уровне, доступном программисту. Используя хитрые методы записи информации, например форматирование дорожки, прерываемое в ходе выполнения, удавалось создавать дискеты с такими характеристиками, которые практически невозможно было повторить даже с помощью специальных контроллеров, например платы Option Board Deluxe, разработанной компанией Central Point Software.

Но в начале 90-х годов появилась программа FDA (Floppy Disk Analyser), разработанная российской компанией Мединком. Эта программа была способна анализировать содержимое дорожек дискеты и с высокой степенью точности создавать копии. Большая часть защищенных дискет копировалась в автоматическом режиме, а для особо сложных случаев был предусмотрен режим ручного управления копированием.

Весьма забавно, что сама программа FDA поставлялась на ключевой дискете, допускающей четыре инсталляции с привязкой к компьютеру.

Однако с появлением операционной системы Windows NT, под которой прямая работа с контроллером жестких дисков из прикладной программы стала невозможной (для этого требовалась установка драйвера), ключевые дискеты практически перестали применяться для защиты от копирования. К тому же, им на смену пришли компакт-диски.

11.2. Привязка к компакт-дискам

Несмотря на внешнюю несхожесть дискет и компакт-дисков, очень многие методы создания некопируемых магнитных носителей были успешно перенесены на оптические диски.

Разумеется, большинство используемых компакт-дисков не допускает перезаписи, а те, что допускают, не позволяют изменять информацию в произвольном месте — можно только дописать новые данные или стереть все, что было записано ранее. Поэтому на компакт-дисках не делают защиты со счетчиком установок.

Однако существует множество способов создать диски, которые не копируются стандартными средствами или для которых существует надежный способ отличить копию от оригинала. Рассмотрим наиболее распространенные из них.

Прежде всего, стоит отметить, что получать доступ к содержимому компакт-диска можно на нескольких уровнях.

Самый высокий уровень — это уровень файловой системы. Данные записываются на диск в определенном формате (например ISO-9660), и драйвер файловой системы компакт-диска (CD-ROM File System, CDFS) отвечает за

то, чтобы представить содержимое диска в виде дерева каталогов и файлов. На этом уровне доступны такие операции, как получение списка файлов, открытие файла с определенным именем и чтение из него информации.

Следующий уровень — уровень секторов. Грубо говоря, на этом уровне диск представляется как последовательность секторов, содержащих полезные данные, и таблица, описывающая содержимое диска (Table Of Contents, TOC). Доступны операции чтения TOC и секторов с заданными номерами.

Самый низкий уровень — уровень команд контроллера. Разные приводы CD-ROM могут иметь различия в доступном наборе команд, но только на этом уровне можно получить самую полную информацию, которую способен выдать привод относительно установленного диска. Использование этого уровня требует разработки драйвера.

11.2.1. Простейшие защиты

Если пользователь пытается сделать копию диска на уровне файловой системы, он сначала копирует все дерево каталогов и файлов на винчестер, а потом с помощью любой программы для создания компакт-дисков записывает диск, содержащий скопированные файлы.

Программа, привязанная к компакт-диску, может проверять метку тома диска, которая не сохраняется при копировании файлов на винчестер. Правда, эту метку можно установить вручную при создании образа нового диска. Так что лучше проверять серийный номер, который выбирается случайным образом при создании диска и не может быть задан пользователем.

Также на какой-нибудь файл можно сделать несколько ссылок из разных директорий. При этом легко добиться того, что суммарный размер файлов, скопированных на жесткий диск, превысит размер компакт-диска.

У какого-нибудь файла в директории компакт-диска можно установить очень большой размер, что не позволит прочитать этот файл, т. к. его данные просто не будут существовать.

Но все эти методы оказываются бессильны, если выполняется копирование диска на уровне секторов, а не на уровне файловой системы. Сейчас посекторное копирование поддерживает почти любая хорошая программа для создания компакт-дисков, например Nero Burning ROM, разработанная компанией Ahead Software AG.

И, разумеется, для борьбы с посекторным копированием применяются другие методы.

11.2.2. Диски большой емкости

Стандартный компакт-диск вмещает 640 Мбайт данных. Однако можно, незначительно изменив параметры диска, уместить на него 700 и даже 800 Мбайт. При этом диск без проблем будет читаться в большинстве приводов CD-ROM.

Даже если никаких специальных проверок не выполняется, для того чтобы сделать точную копию диска большого объема, изготовленного методом штамповки, требуется записываемый диск, вмещающий необходимое количество информации, пишущий привод, способный правильно записать такой диск, и подходящая программа записи. Еще несколько лет назад это могло служить серьезным препятствием, но сейчас найти необходимые болванки и CD-Writer с поддержкой Overburn (запись дисков большой емкости) и хорошую программу для записи дисков не составляет труда.

11.2.3. Отклонение от стандарта записи на диск

Иногда создатели защищенных дисков сознательно идут на нарушение стандарта, описывающего, как и что должно записываться на диск. Драйвер файловой системы использует далеко не всю информацию, которую можно получить о диске, а только то, что необходимо для определения размера диска и доступа к отдельным файлам. А программы посекторного копирования стремятся использовать максимум информации и часто отказываются работать с диском, если встречают противоречивые данные.

Правда, сейчас многие программы позволяют игнорировать некоторые нарушения стандарта и успешно копируют большинство дисков, защищенных таким способом.

Нарушения стандарта имеют еще один негативный аспект: они могут привести к тому, что диск не будет читаться на некоторых компьютерах, а то и вообще вызовет поломку привода CD-ROM.

11.2.4. Физические ошибки на диске

Если диск содержит сознательно внесенные нарушения в области данных, которые приводят к ошибкам чтения, это не обязательно является нарушением стандарта — ошибки могли возникнуть и по естественным причинам, таким как загрязнение или механическое повреждение носителя. Следовательно, все приводы должны правильно отрабатывать ситуации, когда определенный сектор не может быть прочитан. А программа может принимать решение о подлинности диска на основании того, что некоторые строго определенные сектора не читаются.

Программы посекторного копирования часто отказываются продолжать работу с диском, если не могут прочитать очередной сектор, а если и создают новый диск, то на нем непрочитанные с оригинала секторы заполняются нулями или произвольными данными и больше не содержат ошибок.

Но существуют и другие программы, работающие с контроллером напрямую и выполняющие копирование не на уровне логических секторов, а, фактически, на уровне "сырых" данных, которые привод получает с диска. Иногда это называют побитовым копированием.

Наверное, самым популярным инструментом для побитового копирования компакт-дисков была программа CloneCD, разработанная компанией; Elaborate Bytes AG. О программе приходится говорить в прошедшем времени, т. к. на официальной странице CloneCD в Интернете вывешено сообщение о том, что продажи и распространение прекращены. Причина такого решения связана с новым законом о защите авторских прав, но деталей, к сожалению, не приводится.

Однако кроме CloneCD существует много других программ, успешно справляющихся с побитовым копированием практически любых компакт-дисков и создающих копии, принимаемые защитой за оригинал.

Также существуют программы, эмулирующие компакт-диски. Они позволяют использовать заранее сохраненный образ компакт-диска для того, чтобы изображать привод CD-ROM с установленным в нем диском. Многие эмуляторы, например Daemon Tools, умеют передавать не только содержимое диска, но и все ошибки, которые используются для предотвращения копирования и проверки аутентичности компакт-диска.

Однако существуют и системы защиты компакт-дисков, которые долгое время весьма успешно противостояли программам побитового копирования и эмуляторам. Примером такой защиты может служить StarForce.

11.3. Система защиты StarForce Professional

Про StarForce (SF), систему защиты программного обеспечения, распространяемого на дисках CD-ROM, от несанкционированного тиражирования, пока написано не очень много. В основном это информация рекламного характера, исходящая от разработчиков, но попадаются высказывания и тех, кто эту защиту пытался обойти.

На официальном интернет-сайте приводится следующее описание характеристик системы защиты StarForce Professional:

- SF Professional не позволит запустить программный продукт, если компакт-диск идентифицирован как скопированный. Вне зависимости

от того, где и как была сделана копия диска (в домашних условиях или на заводском оборудовании), система определит, что данный диск — нелегальный;

- компакт-диски, защищенные SF Professional, не копируются такими программами, как CloneCD, CDRWin, BlindWrite и им подобными. Защищенные приложения не запускаются на эмуляторах компакт-дисков, к которым относятся Daemon Tools, Virtual CD-ROM и т. п.;
- используя комплект разработчика на этапе создания программного кода, можно значительно усилить защиту приложения против самых эффективных методов взлома;
- для встраивания защиты SF Professional не требуется специального технологического оборудования, нужен только компьютер и доступ на один из серверов StarForce;
- компакт-диски, защищенные SF Professional, максимально совместимы с разнообразными моделями существующих устройств CD/DVD-ROM. Это обусловлено тем, что в SF Professional используется уникальный метод определения подлинности диска без вмешательства в его физическую структуру;
- система защиты использует алфавитно-цифровой 24-значный ключ, который вводится пользователем защищенного программного приложения в процессе эксплуатации только один раз — в момент первого запуска. Ключ будет работать исключительно с дисками данной партии программного обеспечения.

Некоторую полезную информацию можно почерпнуть из интервью с Игорем Павлюком, менеджером по связям с общественностью компании Protection Technology, которая является разработчиком StarForce. В интервью приводятся цитаты, собранные в различных форумах исследователей программ, в том числе зарубежных, где активно происходит обсуждение возможностей взлома или копирования защищенного программного обеспечения.

Так в одном из сообщений на **http://cdfreaks.com** выдвигается призыв уничтожить новую защиту в зародыше (*StarForce Copy protection — Kill the bird in its egg*).

В другом сообщении на том же сайте можно заметить техническое любопытство: как StarForce ухитряется обходить методы побитового копирования, используемые программами типа CloneCD? (*I'm curious how they are able to bypass the 1:1 copy-method that CloneCD and all other burning programs use...*).

Но любопытство быстро сменяется на серьезные опасения, что новая система защиты скоро станет весьма популярной, потому что не требуется специальное оборудование, не существует универсального способа взлома и защищенные диски не копируются с помощью CloneCD (*This StarForce protection system for CD's and CD-R's seems to be very popular soon, because all*

steps in making protected cd's can be done inhouse; also there is no generic crack available and this protection can't be copied by CloneCD).

А в одном сообщении на форуме поддержки Daemon Tools (одного из самых мощных эмуляторов компакт-дисков) утверждается, что сделать копию диска, защищенного StarForce, практически невозможно из-за особенностей защитного механизма *(ft will be nearly impossible to make a backup of StarForce CDs, because of the nature of their protection).* Да и разработчик Daemon Tools подтверждает это утверждение своей репликой о том, что защищенный диск может быть скопирован только на специальную болванку, соответствующую конкретной партии дисков (*What concerns StarForce it is not possible to burn even theoretically with any program or writer, unless you get special media, which can be differentfor each title or even party of CDs of same title. So forget if).*

Однако, как известно, непробиваемых защит не бывает, что подтверждают, например, статьи на **www.reversing.net**, в которых рассказывается о методах получения расшифрованной версии EXE-файла, работоспособной без оригинального компакт-диска.

В форуме Daemon Tools один человек утверждал, что ему удалось сделать копию диска, которая опознается как оригинал в 9 случаях из 10. А в другом сообщении разработчик Daemon Tools практически обещает реализовать эмуляцию дисков, защищенных StarForce *(You have to wait until dumping programs appear that can dump it correctly. Most likely FantomCD will be one of the programs capable to produce such images (MDS format). Beta version of Daemon already works successfully with mounted StarForce images — so the question is in images only).*

На фоне всего вышеизложенного задача оценить, насколько надежной является защита StarForce и что она собой представляет в действительности, показалась довольно интересной. Ниже приводятся результаты научно-исследовательской работы, выполненной кафедрой "Информационная безопасность" (ИУ-8) МГТУ им. Н. Э. Баумана по соглашению с компанией Protection Technology. Представители Protection Technology не возражали против публикации этих результатов. В качестве экспериментального образца использовалась игра "Heroes of Might and Magic IV", защищенная StarForce 2.0.

11.3.1. Общая характеристика защиты

Защитные механизмы, работающие на компьютере конечного пользователя защищенного диска, можно условно разделить на две части. К первой части отнесем все способы противодействия исследованию защищенной программы и приведению исполняемых файлов к состоянию, в котором они будут способны работать без оригинального компакт-диска. Во второй части окажется непосредственно механизм проверки подлинности компакт-диска.

Исследование средств защиты исполняемых модулей от отладки и снятия правильно работающего дампа — занятие неблагодарное. Для грамотно защищенной программы, с умом использующей все возможности, предоставляемые защитой, процесс восстановления исполняемого модуля доступен только высококлассным специалистам и практически не поддается автоматизации. То есть для снятия защиты с каждой новой программы потребуется большую часть исследований проводить сначала. Да и полной гарантии стопроцентной работоспособности получить не удастся. Фрагменты защиты могут быть вставлены в труднодостижимые места. Например, какая-нибудь проверка вполне может выполняться только в седьмой миссии многоуровневой игры, до которой невозможно добраться быстрее, чем за трое суток непрерывных сражений! Так что оставим снятие защиты через восстановление исполняемых модулей фанатикам исследований программ и перейдем к рассмотрению части защиты, связанной с проверкой аутентичности компакт-диска.

Как утверждают разработчики StarForce, при изготовлении защищенных дисков не требуется никакое специальное оборудование, позволяющее наносить лазерные метки или какие-либо иные повреждения поверхности компакт-диска. Да и современные программы побитового копирования дисков, такие как CloneCD или BlindRead/BlindWrite, способны настолько точно воссоздавать все ошибки, что защита оказывается неспособна отличить оригинал от копии. Однако практика показывает, что в подавляющем большинстве случаев копия диска, защищенного StarForce, не опознается как оригинальный диск, какой бы программой ни выполнялось копирование.

Так как же StarForce опознает оригинальный диск? Правильный ответ на этот вопрос знают только разработчики, однако в форуме поддержки Daemon Tools можно найти высказывание, что StarForce использует информацию об углах между секторами и метод получения этой информации совместим с 99.9 % приводов CD-ROM *(StarForce uses angle info and the method of retrieving this makes it 99.9 % compatible with any CD-ROM)*.

Попробуем проверить гипотезу об определении аутентичности диска путем измерения его угловых характеристик. Для этого смоделируем процессы, происходящие при чтении диска.

11.3.2. Модель задержек при чтении информации с компакт-диска

В популярных источниках легко найти описание характеристик звукового компакт-диска.

Компакт-диск (КД)

КД имеет диаметр 120 мм и центральное посадочное отверстие диаметром 15 мм. Зона записи звука заключена в кольце с внутренним диаметром 50 мм и наружным — 116 мм. Вне ее находится зона, содержащая вспомогательную информацию, которая позволяет автоматизировать процесс воспроизведения. Сигнал записан на дорожке, расположенной на КД в виде спирали. Шаг витков спирали 1.6 мкм, т. е. поперечная плотность записи 625 дорожек/мм. Всего дорожка образует на КД 20 000 витков общей протяженностью 5 км и начинается не у наружной границы зоны записи, как на обычных грампластинках, а у внутренней.

Все вышесказанное справедливо и для компакт-дисков, на которых записаны данные. Спираль разбивается на последовательно идущие сектора, длиной 2352 байт каждый (16-байтовый заголовок, 2048-байтовая область данных и 288-байтовая зона коррекции ошибок). Также известно, что линейная плотность информации вдоль спирали является постоянной на всем диске.

Для дальнейших рассуждений примем, что расстояние между дорожками (1.6 мкм) одинаково на любых компакт-дисках, а длина сегмента спирали, принадлежащего одному сектору, является постоянной для конкретного экземпляра диска. Размеры зоны записи (внутренний и внешний радиусы) и полезная емкость носителя могут варьироваться от одного диска к другому. Так современные матрицы для записи КД имеют емкость от 650 до 800 Мбайт.

Положение на диске сектора с любым номером однозначно описывается двумя характеристиками диска:

R_{inner} — расстояние от центра диска, на котором начинается нулевой сектор спирали;

L_{sect} — длина сегмента спирали, соответствующая одному сектору.

Выведем формулы, необходимые для определения точного положения сектора на диске по его номеру. Достаточно вспомнить школьный курс математики, потребуются лишь формула вычисления длины окружности и навыки по выполнению простейших арифметических операций.

Число витков спирали N с поперечной плотностью D витков/мм от радиуса R_1 до радиуса R_2 определяется формулой:

$$N = (R_2 - R_1) * D$$

Длина спирали L в том же диапазоне радиусов выражается как:

$$L = n * (R_2 + R_1) * N = \pi * (R_2 + R;J * (R_2 - R_1) * D = n * (R_2^2 - R,^2) * D$$

Расстояние L_i от начала спирали до i-ого сектора будет равно:

$$L_i = i * L_{sect} = \pi * (R_i^2 - R_{inner}^2) * D$$

Радиус R_i, на котором начинается i-ый сектор, определяется как:

$$R_i = Sqrt\ (i * L_{sect} / D / \pi + R_{inner}^2)$$

Число витков N_i от начала спирали до i-ого сектора вычисляется по формуле:

$$N_i = (R_i - R_{inner}) * D = (Sqrt\ (i * L_{sect} / D / \pi + R_{inner}^2) - R_{inner}) * D \quad (1)$$

Целая часть N_i задает номер витка, а дробная часть — угловое положение сектора.

Теперь перейдем к физическим характеристикам привода.

В качестве базового тезиса при разработке компакт-дисков использовалась идея о постоянной линейной плотности записанных данных, а значит, и постоянной линейной скорости чтения диска. Но из-за того что длина витка спирали зависит от радиуса, для обеспечения постоянной линейной скорости чтения угловая скорость вращения диска должна быть переменной. И в первых приводах скорость вращения диска изменялась примерно от 500 оборотов в минуту на внутренних витках спирали до 200 оборотов в минуту на внешних, более длинных витках. Однако в настоящее время существуют многоскоростные приводы, у которых угловая скорость вращения диска является постоянной, а линейная скорость чтения растет при переходе к внешним виткам спирали. И, судя по всему, таких приводов большинство, т. к. ограничения на повышение скорости передачи информации, читаемой с компакт-диска, накладываются не столько интерфейсом между приводом и оперативной памятью компьютера, сколько механическими свойствами самого привода, например значительными вибрациями на больших скоростях вращения. И практически нет разумных поводов для снижения скорости вращения при чтении информации с внешних витков спирали. Таким образом, будем исходить из того, что привод, с которым мы имеем дело, имеет постоянную угловую скорость вращения диска и двигатель привода выключается только по истечении некоторого значительного периода времени, на протяжении которого не было ни одного обращения.

Что происходит после того, как пользовательская программа инициировала команду чтения какого-то сектора диска? Грубо последовательность действий может быть описана примерно следующим образом. Сначала запрос на чтение обрабатывается драйверами операционной системы, которые передают этот запрос приводу. Привод осуществляет позиционирование головки, дожидается, пока диск не повернется до начала сектора, читает данные с диска и передает их в память, а потом присылает извещение о том, что операция чтения завершилась. Дальше происходит окончательная обработка

запроса драйверами операционной системы, и прочитанный сектор или несколько последовательных секторов передаются пользовательской программе.

Точно определить, какое время занимает выполнение того или иного шага приведенной выше схемы, не представляется возможным. Однако если предположить, что длительность постобработки драйверами операционной системы не зависит от номера читаемого сектора, а привод извещает о выполнении операции сразу по окончании чтения последнего из требуемых секторов, то временная задержка между двумя любыми операциями чтения должна с незначительными допущениями укладываться в следующую формулу:

$$Ty = (n + fract(N_j) - fract(N_i)) * P, \qquad (2)$$

где:

- i, j — номер сектора, следующего за последним прочитанным во время первого или второго запроса сектором;
- T_{ij} — задержка между окончаниями выполнения запросов;
- N_i, N_j — положения /-ого и j-ого сектора на спирали, вычисленные по формуле (1);
- $fract(x)$ — дробная часть x;
- P — период вращения диска (время, за которое происходит один полный оборот);
- n — произвольное целое число.

То есть задержка состоит из времени, необходимого для нескольких полных оборотов диска, и времени на поворот диска от углового положения *fract (NJ* до углового положения *fract (Nj)*.

11.3.3. Как StarForce проверяет диск

Проверка подлинности диска состоит из нескольких этапов. Сначала читается информация о диске, установленном в приводе, и проверяется его метка тома. Затем выполняется 8 запросов на чтение случайных одиночных секторов с номерами в диапазоне от 1 до 65 536. Результаты чтения никак не используются, и, скорее всего, эти действия нужны для разгона диска до номинальной скорости вращения. Затем еще раз читается (но уже не проверяется) информация о диске. Все перечисленное выше проходит через драйвер файловой системы CDFS, никак не защищено от анализа и, следовательно, наверняка не влияет на процесс аутентификации.

Все остальные обращения к диску идут на более низком уровне. В той версии StarForce, анализ которой проводится, обращения адресовались драйве-

ру устройства Cdrom и представляли собой SCSI-команды. Последовательность этих команд такова.

1. Чтение содержания диска (Table Of Content, TOC).
2. Чтение одиночных секторов с номерами 16, 17, 17 (дважды читается 17-ый сектор).
3. Чтение одиночных секторов с номерами 173117, 173099, 173081, 173063, 173045, 173027, 173009, 172991, 172973.
4. Чтение случайных 17 блоков по 8 секторов с номерами первого читаемого сектора в диапазоне примерно от 168100 до 173200.
5. SCSI-команда с кодом OxBB, описание которой не удалось найти в документации, но которая, скорее всего, отвечает за управление скоростью вращения привода.
6. Чтение одиночного сектора с номером 173117.

Причем если с первой попытки диск не опознан как оригинальный, то шаги 3 и 4 повторяются в цикле. Значит, после выполнения шага 4 вся информация, необходимая для аутентификации диска, уже получена.

Попробуем разобраться, зачем может потребоваться каждый из шагов.

Чтение TOC, скорее всего, требуется для определения номера сектора, с которого начинается последняя сессия мультисессионного диска. Так как сессия всего одна, то в 16 и 17 секторах как раз и хранятся описания структуры тома (метка тома, количество секторов, адрес директории диска и т. д.). А повторное чтение сектора 17, скорее всего, используется для того, чтобы примерно оценить порядок времени, затрачиваемого на один оборот диска. Разница времени между двумя чтениями одного сектора должна быть кратна длительности оборота диска.

В последовательности номеров секторов 173117, 173099, 173081, 173063, 173045, 173027, 173009, 172991, 172973 легко усматривается закономерность — каждое следующее значение на 18 меньше предыдущего. Число 18 тоже явно не случайное — на том радиусе диска, где размещаются сектора с указанными номерами, на один виток спирали помещается примерно 18 секторов. А чтение секторов в порядке убывания номера с большой вероятностью используется для того, чтобы предотвратить чтение с предупреждением, когда привод считывает во внутренний буфер не только заданные сектора, но и несколько последующих, на случай если данные читаются последовательно.

Получив значения восьми интервалов (между девятью операциями чтения) и зная длительность и периодов обращения диска (полученную повторным чтением сектора), можно с большой точностью определить скорость вращения диска.

А дальше выполняется 17 чтений блоков со случайными номерами с целью измерения 16 интервалов времени. Если все интервалы хорошо (с малыми отклонениями) укладываются в формулу (2), то диск признается подлинным. Если же отклонения от ожидаемых величин превышают некоторое пороговое значение, то проводится повторное вычисление скорости вращения и повторное измерение задержек между чтением блоков по 8 секторов.

11.3.4. Способ обхода защиты

Чтобы заставить StarForce поверить, что в приводе стоит оригинальный диск, надо совсем не много: чтобы задержки между чтениями соответствовали ожидаемым. А для этого необходимо знать точные характеристики диска: радиус, на котором начинается спираль, и размер сектора. Для определения этих величин можно провести те же самые измерения, что проводит StarForce при проверке диска, а затем варьировать начальный радиус и размер сектора, пока не будут найдены оптимальные значения. Критерием оптимальности, например, может служить сумма отклонений разностей углов, вычисленных по формуле (1), и углов, полученных из замеренных интервалов времени по формуле, обратной (2).

Современное оборудование (во всяком случае, оборудование бытового класса) действительно не позволяет создавать копии защищенного диска, но написание эмулятора, способного обмануть StarForce, не представляет сверхсложной задачи. Достаточно перехватывать обращения к драйверу CD-ROM и в случае, если выполняется команда чтения, делать временную задержку, какую мог бы иметь оригинальный диск, и только после этого возвращать управление вызывающей программе.

В качестве практической демонстрации возможности эмуляции был разработан драйвер, функционирующий под операционной системой Windows 2000 и выполняющий описанные выше действия. Когда драйвер загружен, StarForce оказывается не в состоянии отличить подделку от оригинала. Игра стабильно запускается практически с любой копии оригинального диска, с виртуального диска, созданного программой Daemon Tools, и даже с дисков, которые похожи на оригинальный только тем, что имеют правильную метку тома и размер области данных не менее 350 Мбайт, чтобы существовали сектора с запрашиваемыми номерами.

11.3.5. Резюме по защите StarForce

StarForce, несомненно, является неординарной защитой. Ее исключительность заключается хотя бы в том, что до сих пор не существует надежного способа быстро создавать работоспособные копии защищенных дисков.

Однако проведенное исследование показывает, что для эмуляции защищенного диска нужно совсем немного.

Примерно через 3 месяца после выполнения работы, результаты которой были приведены ранее, компания Protection Technology объявила о выпуске следующей версии своей системы защиты — StarForce Professional 3.0. Разработчики утверждают, что одно из многочисленных улучшений заключается как раз в усилении противодействия эмуляции компакт-дисков.

Кстати, вскоре после появления StarForce 3.0 буквально в течение одного месяца авторы как минимум трех эмуляторов компакт-дисков объявили о том, что новые версии их программ способны эмулировать диски, защищенные StarForce версий 1 и 2. С тех пор прошло больше года, но поддержка StarForce 3.0 так и не появилась ни в одном из эмуляторов. Так что по состоянию на сегодняшний день, компакт-диски, защищенные при помощи StarForce, продолжают оставаться устойчивыми к взлому.

ıва 12



паратные ключи защиты

иного лет на рынке средств защиты программ от несанкционирован- тиражирования присутствуют так называемые аппаратные ключи защиты ;les). Разумеется, компании, продающие такие устройства, представля- если не как панацею, то уж как надежное средство противодействия ьютерному пиратству. Но насколько серьезным препятствием могут ть аппаратные ключи?

. Классификация ключей

)атные ключи защиты можно пытаться классифицировать по несколь- ризнакам.

рассматривать возможные типы подключения, то бывают, например, і на порт принтера (LPT), последовательный порт (COM), USB-порт и і, подключаемые к специальной плате, вставляемой внутрь компьютера.

ю при сравнении ключей анализировать удобство и функциональность ствующего программного обеспечения. Например, для некоторых се- в аппаратных ключей разработаны автоматические протекторы, позво- ие защитить программу "за один клик", а для некоторых такие протек- отсутствуют.

іеленный интерес представляет список языков программирования, для ых разработчик ключей предоставил библиотеки и примеры. Поддерж- ыков (доступ к API ключа из определенной среды) нужна для того, программист смог более эффективно использовать ключ для защиты)атываемой программы.

і также список аппаратных платформ и операционных систем, для ко- поддерживается интерфейс с ключом.

"орых может заинтересовать применимость ключа для сетевого лицен- ания программного обеспечения.

Однако все сказанное о ключах относится скорее к маркетингу, чем к защите информации. Для защиты не важно, какого цвета корпус у ключа и на каком языке можно читать документацию. А по-настоящему важно только то, что в ключе является секретным и неповторимым и способно ли это "нечто" обеспечить необходимый уровень защиты.

Поэтому в дальнейшем ключи рассматриваются исключительно как аппаратные устройства, работающие в определенных условиях и имеющие некоторую функциональность. Полезными признаются только те функции, которые невозможно реализовать чисто программными средствами и для которых не существует эффективной атаки.

Будем исходить из предположения, что у противника есть физический доступ к ключу, а основная задача заключается в том, чтобы за разумное время получить копию программы, функционирующую в отсутствие ключа точно так же, как при его наличии.

Рассматривать атаки на систему, в которой не хватает некоторых узлов, необходимых для работы, особого смысла нет — если зашифровать программу и не сообщить противнику ключ шифрования, легко получить высокую стойкость и без применения аппаратных ключей. Только это уже нельзя называть защитой от копирования.

12.2. Модификация кода и эмуляция

Для того чтобы заставить программу работать так, как она работала бы с ключом, можно или внести исправления в программу, или эмулировать наличие ключа. Модификация программы, как правило, возможна лишь в тех случаях, когда ответы, полученные от ключа, просто проверяются, но не являются математически необходимыми для обеспечения работоспособности программы. Но это значит, что ключ, по большому счету, не требуется для достижения полной функциональности. Такое случается, когда программа не использует всех возможностей ключа или когда возможности ключа очень ограничены.

При эмуляции никакого воздействия на программу не происходит, т. е., например, не нарушается контрольная сумма исполняемых модулей. И полный эмулятор, если его удается построить, просто повторяет все поведение реального ключа.

Не вдаваясь очень глубоко в технические подробности, будем исходить из предположения, что у противника есть следующие возможности:

- перехватывать все обращения к ключу;
- протоколировать и анализировать эти обращения;

- посылать запросы к ключу;
- получать ответы от ключа;
- протоколировать и анализировать эти ответы;
- посылать ответы от имени ключа.

Такие широкие возможности противника можно объяснить тем, что в его распоряжении есть вся та информация, какая есть и у программиста, защищающего программу с помощью аппаратного ключа. То есть противник имеет доступ ко всем открытым интерфейсам, документации, драйверам и может их анализировать на практике с привлечением любых средств. Следовательно, можно предположить, что противник со временем научится полностью контролировать протокол, по которому происходит обмен информацией между прикладной программой и ключом. Контроль может осуществляться на любом уровне, но чаще всего запросы перехватываются при передаче данных между программой и драйвером ключа.

Однако стоит учитывать, что возможность эмуляции еще не означает, что противник способен вычислять правильные ответы на любые запросы, которые посылает ключу программа.

12.3. Ключи с памятью

Это, наверное, самый простой тип ключей. Ключи с памятью имеют определенное число ячеек, из которых разрешено считывание. В некоторые из этих ячеек также может производиться запись. Обычно в ячейках, недоступных для записи, хранится уникальный идентификатор ключа.

Когда-то давно существовали ключи, в которых перезаписываемой памяти не было вообще, а программисту для считывания был доступен только идентификатор ключа. Но очевидно, что на ключах с такой функциональностью построить серьезную защиту просто невозможно.

Правда, и ключи с памятью не способны противостоять эмуляции. Достаточно один раз прочитать всю память и сохранить ее в эмуляторе. После этого правильно эмулировать ответы на все запросы к ключу не составит большого труда.

Таким образом, аппаратные ключи с памятью в заданных условиях не способны дать никаких преимуществ по сравнению с чисто программными системами.

12.4. Ключи с неизвестным алгоритмом

Многие современные аппаратные ключи содержат секретную функцию преобразования данных, на которой и основывается секретность ключа. Иногда

программисту предоставляется возможность выбрать константы, являющиеся параметрами преобразования, но сам алгоритм остается неизвестным.

Проверка наличия ключа должна выполняться следующим образом. При разработке защиты программист делает несколько запросов к алгоритму и запоминает полученные ответы. Эти ответы в какой-то форме кодируются в программе. Во время выполнения программа повторяет те же запросы и сравнивает полученные ответы с сохраненными значениями. Если обнаруживается несовпадение, значит, программа получает ответ не от оригинального ключа.

Эта схема имеет один существенный недостаток. Так как защищенная программа имеет конечный размер, то количество правильных ответов, которые она может хранить, также является конечным. А это значит, что существует возможность построения табличного эмулятора, который будет знать правильные ответы на все запросы, результат которых может проверить программа.

В рекомендациях по защите программ с помощью аппаратных ключей даются советы, как сделать фиктивные запросы со случайными данными так, чтобы затруднить построение эмулятора. Однако если программа при запуске делает 100 запросов, результат которых может быть проверен, и 100 случайных запросов, результат которых не проверяется, то, запустив программу 10 раз, очень легко выделить действительные запросы, повторившиеся 10 раз, и отсечь все фиктивные, встретившиеся по 1–2 раза.

Конечно, не стоит всегда проверять наличие ключа выполнением одной и той же серии запросов с проверкой. Лучше выполнять проверки в разных частях программы и в разное время. Это может значительно усложнить сбор статистики для отсечения фиктивных запросов.

Но не стоит забывать, что противник может проанализировать программу и попытаться в дизассемблере найти все обращения к ключу. Это поможет ему выяснить, ответы на какие из запросов проверяются, и построить компактную таблицу для эмуляции.

Так что ключи с неизвестным алгоритмом могут затруднить, но не могут предотвратить построение эмулятора для конкретной версии конкретной программы. Зато при переходе к новой версии, если перечень проверяемых программой ответов на запросы будет изменен, противнику придется заново выполнять сбор статистики или анализ программы.

12.5. Атрибуты алгоритмов

В некоторых ключах алгоритму могут сопутствовать дополнительные атрибуты. Так, например, в ключах Sentinel SuperPro алгоритм может быть защищен паролем и начинает работать только после того, как будет выполне-

на активация, в ходе которой правильный пароль должен быть передан ключу.

Активация позволяет разработчику предусмотреть возможность изменения функциональности ключа на стороне пользователя. То есть программа может иметь несколько версий (например базовую, расширенную и профессиональную), и в ключе изначально активированы только те алгоритмы, которые необходимы для функционирования базовой версии. Если же пользователь решит перейти к более полной версии, разработчик пришлет ему инструкции по активации алгоритмов, соответствующих расширенной или профессиональной версии.

Однако все достоинства алгоритмов, активируемых по паролю, опираются на секретность пароля, а не на свойства аппаратного ключа. Следовательно, аналогичная защита может быть реализована чисто программными средствами.

Другой тип атрибутов алгоритмов, поддерживаемых ключами Sentinel SuperPro, — это счетчики. С активным алгоритмом может быть связан счетчик, изначально имеющий ненулевое значение. Программа при каждом запуске (или выполнении определенной операции, например при экспорте данных) вызывает специальную функцию API-ключа, уменьшающую значение счетчика на единицу. Как только счетчик принимает нулевое значение, алгоритм деактивируется и перестает работать.

Однако данная схема не способна помешать применению эмулятора. Противник может перехватывать и предотвращать все попытки уменьшения значения счетчика. Следовательно, алгоритм никогда не будет деактивирован, и в распоряжении противника будет неограниченное время для сбора данных, необходимых для табличной эмуляции.

Противостоять эмуляции может счетчик, значение которого уменьшается при каждом обращении к алгоритму. Но в этом случае возникает опасность, что из-за сбоев в работе программы или операционной системы иногда значение счетчика будет уменьшаться без совершения программой полезных Действий. Причина проблемы в том, что обращение к алгоритму должно производиться до того, как программа совершит полезную работу, а счетчик Должен уменьшаться только в том случае, если работа выполнена успешно. Но автоматическое уменьшение счетчика при обращении к алгоритму такую Функциональность не обеспечивает — количество оставшихся попыток Уменьшается независимо от успеха выполненной операции.

12.6. Ключи с таймером

Некоторые производители аппаратных ключей предлагают модели, имеющие встроенный таймер. Но для того чтобы таймер мог работать в то время,

когда ключ не подключен к компьютеру, необходим встроенный источник питания. Среднее время жизни батареи, питающей таймер, составляет 4 года, и после ее разрядки ключ перестанет правильно функционировать. Возможно, именно из-за сравнительно короткого времени жизни ключи с таймером применяются довольно редко.

Но как таймер может помочь усилить защищенность?

Ключи HASP Time предоставляют возможность узнавать текущее время, установленное на встроенных в ключ часах. И защищенная программа может использовать ключ для того, чтобы отследить окончание тестового периода. Но очевидно, что эмулятор позволяет возвращать любые показания таймера, т. е. аппаратная часть никак не повышает стойкость защиты.

Хорошей комбинацией является алгоритм, связанный с таймером. Если алгоритм может быть деактивирован в определенный день и час, очень легко будет реализовывать демонстрационные версии программ, ограниченные по времени.

Но, к сожалению, ни один из двух самых популярных в России разработчиков аппаратных ключей не предоставляет такой возможности. Ключи HASP, производимые компанией Aladdin, не поддерживают активацию и деактивацию алгоритмов. А ключи Sentinel SuperPro, разработанные в Rainbow Technologies, не содержат таймера.

12.7. Ключи с известным алгоритмом

В некоторых ключах программисту, реализующему защиту, предоставляется возможность выбрать из множества возможных преобразований данных, реализуемых ключом, одно конкретное преобразование. Причем подразумевается, что программист знает все детали выбранного преобразования и может повторить обратное преобразование в чисто программной системе.

Например, аппаратный ключ реализует симметричный алгоритм шифрования, а программист имеет возможность выбирать используемый ключ шифрования. Разумеется, ни у кого не должно быть возможности прочитать значение ключа шифрования из аппаратного ключа.

В такой схеме программа может передавать данные на вход аппаратного ключа и получать в ответ результат шифрования на выбранном ключе. Но тут возникает дилемма. Если в программе отсутствует ключ шифрования, то возвращаемые данные можно проверять только табличным способом, а значит, в ограниченном объеме. Фактически имеем аппаратный ключ с неизвестным программе алгоритмом. Если же ключ шифрования известен программе, то можно проверить правильность обработки любого объема данных, но при этом существует возможность извлечения ключа шифрова-

ния и построения эмулятора. А если такая возможность существует, противник обязательно попытается ею воспользоваться.

Так что аппаратное выполнение симметричного алгоритма шифрования с известным ключом не дает ничего нового с точки зрения защиты. Но есть еще и асимметричные алгоритмы.

Когда ключ реализует асимметричный алгоритм шифрования, программисту не обязательно знать используемый секретный ключ. Даже можно сказать, что отсутствие возможности создать программную копию аппаратного асимметричного шифрующего устройства не сужает, а расширяет область возможных применений, т. к. сокращается перечень возможных способов компрометации секретного ключа. В любом случае, для проверки того, что аппаратный ключ присутствует и правильно выполняет вычисления, достаточно знать открытый ключ.

Эта схема не может быть обойдена только эмуляцией, т. к. для построения полного эмулятора требуется по открытому ключу шифрования вычислить секретный ключ. А это математически сложная задача, не имеющая эффективного решения.

Но у противника остается возможность подмены открытого ключа в программе, и если такая подмена пройдет незамеченной, построить программный эмулятор не составит труда. Так что асимметричные алгоритмы, реализованные на аппаратном уровне, способны обеспечить некопируемость защищенной программы, но только в том случае, если удастся предотвратить подмену открытого ключа шифрования.

12.8. Ключи с программируемым алгоритмом

Очень интересным решением с точки зрения стойкости защиты являются аппаратные ключи, в которых может быть реализован произвольный алгоритм. Сложность алгоритма ограничивается только объемом памяти и системой команд ключа.

В этом случае для защиты программы важная часть вычислений переносится в ключ, и у противника не будет возможности запротоколировать правильные ответы на все запросы или восстановить алгоритм по функции проверки. Ведь проверка, как таковая, может вообще не выполняться — результаты, возвращаемые ключом, являются промежуточными величинами в вычислении какой-то сложной функции, а подаваемые на вход значения зависят не от программы, а от обрабатываемых данных.

Главное — это реализовать в ключе такую функцию, чтобы противник не смог по контексту догадаться, какие именно операции производятся в ключе.

12.9. Что происходит на практике

На практике в подавляющем большинстве случаев программисты не используют все возможности, предоставляемые аппаратными ключами. Так, например, очень часто в алгоритмических ключах с памятью используется только память, не говоря уже о случаях, когда все проверки наличия ключа производятся в одной функции, которая возвращает результат в виде логического значения. И для получения полнофункциональной версии программы даже не требуется ключ— достаточно исправить функцию проверки, чтобы она всегда возвращала состояние, соответствующее наличию ключа.

Некоторые ключи (например Sentinel SuperPro) имеют довольно сложную систему разграничения доступа. Ключи Sentinel SuperPro поддерживают пароли для активации алгоритмов, выбираемые при программировании, и раздельные пароли для записи и перезаписи, одинаковые для всех ключей одной серии, поставляемых одному разработчику. И очень часто в теле программы оказывается пароль перезаписи, который позволяет противнику перепрограммировать ключ по своему усмотрению.

Ключи HASP Time, напротив, не имеют разграничения доступа — для того чтобы изменить время в ключе, нужно знать те же самые пароли, которые используются для чтения времени. То есть противнику для продления периода работоспособности программы, ограниченной по времени, достаточно отвести назад часы в ключе, и ничто не мешает ему это сделать.

Некоторые неизвестные алгоритмы, реачизованные в ключах, были подвергнуты анализу. Так был восстановлен алгоритм функции `SeedCode`, используемой в ключах HASP. А по некоторым сообщениям в Интернете, не являются больше секретными и алгоритмы, реализуемые ключами Sentinel SuperPro, и даже новые алгоритмы кодирования и декодирования в ключах HASP4.

Ключи с асимметричными алгоритмами выпускаются уже многими производителями, но позиционируются как устройства для идентификации и аутентификации, а не для защиты информации.

А ключи с программируемым алгоритмом, напротив, не выпускаются крупнейшими производителями ключей и, следовательно, практически не применяются. Возможно, это обусловлено тем, что они получаются слишком дорогими или слишком сложными для использования.

12.10. Выводы о полезности аппаратных ключей

Утверждение, что аппаратные ключи способны остановить компьютерное пиратство, является мифом, многие годы распространяемым производите-

лями ключей. Для хорошо подготовленного противника ключ редко является серьезным препятствием.

К тому же, часто программисты слепо доверяют автоматизированным средствам защиты, поставляемым в составе SDK-ключа, и не прикладывают самостоятельных усилий для усиления защиты. Обещания производителей создают иллюзию защищенности, но на самом деле практически для всех автоматизированных средств защиты давно разработаны эффективные способы нейтрализации защитных механизмов.

Большая часть защитных механизмов, применяемых в современных ключах, оказывается работоспособной только в предположении, что противник не сможет обеспечить эмуляцию ключа, т. е. реализуются на программном уровне. Следовательно, почти всегда тот же уровень защиты может быть достигнут без привлечения аппаратных средств.

ıава13



спользование тесных защит

ıим из популярных способов защиты программ является использование называемых протекторов — программных инструментов, предназначен-. для зашиты других программ.

ıчно сценарий установки защиты следующий. Разработчик создает про-ıмный продукт с использованием некоторых программных средств: ви-ıьных сред, компиляторов и т. д. После того как получен работающий олняемый файл, этот файл обрабатывается с помощью программы-тектора и создается новый исполняемый файл, в котором реализованы аторые средства защиты.

.1.Какуюзащиту еспечивают протекторы

ıтекторы, прежде всего, защищают программу от исследования. Исследо-> можно различные области программы, но наиболее часто проводится ıедование кода, причем с совершенно разными целями. Исследование а вируса может проводиться с целью определения методов заражения и ıаботки вакцины. Исследование кода операционной системы помогает ɔдить уязвимости, а также писать приложения, взаимодействующие ıерационной системой на более низком уровне. Программы исследуются елью обнаружения недокументированных возможностей, а иногда для :тановления алгоритма, по которому программа функционирует. Суще-'ют и другие причины для исследований.

асти ресурсов и данных также могут содержать некоторую интересную ɔормацию, поэтому часто защите подвергают не только код программы, ı данные с ресурсами.

[ако защищать абсолютно все ресурсы не совсем правильно. Дело в том, основная часть ресурсов должна быть доступна только в момент выпол-ия программы, и такие ресурсы можно безбоязненно защищать. Но есть

некоторое количество ресурсов, например информация о версии программы и ее иконка, которые могут использоваться операционной системой тогда, когда программа не запущена. И эти ресурсы в защищенной программе должны оставаться в открытом виде.

То же самое относится и к некоторым служебным структурам данных, хранящимся внутри программы и использующимся в процессе загрузки. Если эти структуры будут недоступны операционной системе, защищенную программу не удастся запустить.

Многие протекторы содержат средства, позволяющие создавать версии с ограничениями. Например, защищенная программа может прекратить работать через заданный промежуток времени, если не будет введен правильный регистрационный код или до ввода кода будет регулярно появляться окно с напоминанием о том, что программа не зарегистрирована, и с предложением приобрести лицензию.

Наиболее продвинутые протекторы имеют программные интерфейсы (API), доступные из защищаемой программы и позволяющие более четко контролировать процесс ее выполнения. Очень часто API используется для динамической разблокировки фрагментов кода, которые должны быть доступны только в зарегистрированной версии.

13.2. Как работают протекторы

Для того чтобы защитить исполняемый файл, протектор должен каким-то образом преобразовать его содержимое и добавить свой код, отвечающий за правильную загрузку измененной программы в память.

Практически для всех форматов исполняемых файлов были разработаны алгоритмы, позволяющие добавлять новый код таким образом, чтобы он выполнялся до основной программы, не нарушая при этом ее функциональности. Скорее всего, основные исследования в этой области были выполнены авторами вирусов, т. к. добавление тела вируса к программе является одним из основных методов заражения.

Код, данные и. ресурсы обычно защищаются с помощью шифрования. Используемый алгоритм не обязательно должен быть криптографически стойким, т. к. ключ шифрования все равно невозможно сохранить в абсолютной тайне. Очень часто до шифрования применяется сжатие данных, что позволяет компенсировать увеличение размера исполняемого файла, происходящее вследствие добавления кода протектора. А иногда результирующий защищенный файл даже уменьшается в размере по сравнению с исходным файлом.

При запуске защищенной программы управление сразу получает код протектора, который выполняет предусмотренные проверки и расшифровывает

в памяти все необходимые области, а также проводит настройку таблицы адресов импортируемых функций. После успешного завершения процедуры настройки протектор передает управление на оригинальную точку входа (Original Entry Point, OEP) и начинается выполнение основной программы.

Проверки, выполняемые протектором до начала работы программы, могут быть разного рода. Это могут быть проверки, касающиеся наличия лицензии (чтобы без лицензии программа просто не запускалась) или сравнения текущей даты со значением, после которого программа должна перестать работать, а также попытки определить наличие запущенного отладчика.

13.3. Сценарии атаки

Итак, чаще всего защищенная программа отличается от незащищенной по следующим параметрам:

- код самой программы зашифрован;
- адрес оригинальной точки входа известен только протектору;
- основная часть ресурсов зашифрована;
- основная часть данных зашифрована;
- таблицы импорта недоступны (настройку импорта выполняет сам протектор, и только он знает, какие функции должны быть импортированы);
- присутствует код протектора.

Для того чтобы обезвредить (удалить) протектор и реконструировать программу к виду, максимально близкому к незащищенному, человеку, который пытается снять защиту, необходимо решить следующие задачи:

- получить расшифрованный код программы;
- найти адрес оригинальной точки входа;
- получить расшифрованные ресурсы;
- получить расшифрованные данные;
- определить все импортируемые программой функции и восстановить таблицы импорта;
- удалить код протектора.

Для большинства существующих протекторов разными людьми в разное время были разработаны эвристики, позволяющие успешно решить все или почти все эти задачи.

Так, например, код обычной программы не должен изменяться в процессе выполнения. Следовательно, после того, как протектор передал управление защищенной программе, и до ее завершения код представлен в памяти

в таком же виде, как и у незащищенной программы. То есть для восстановления оригинальной секции кода достаточно извлечь ее из памяти после того, как программа была запущена. В современных версиях Windows для этого очень хорошо подходит стандартная функция Win32 API, носящая название ReadProcessMemory.

Найти адрес оригинальной точки входа несколько сложнее. Но, имея в распоряжении расшифрованную секцию кода, можно получить очень много информации, позволяющей эффективно решить эту задачу. По секции кода довольно легко можно определить, в какой среде разработки создана программа и каким компилятором она собрана. А наличие такой информации значительно упрощает отыскание оригинальной точки входа. Так, например, характерной особенностью программ, собранных с помощью Borland C++ Builder, является то, что точка входа находится в самом начале секции кода. А точка входа в программы, собранные в Borland Delphi, напротив, соответствует началу функции, которая расположена в самом конце секции кода. Также в графических (не консольных) приложениях одной из первых вызываемых функций Win32 API является GetModuleHandle, т. к. значение, возвращаемое этой функцией, должно быть передано в функцию WinMain, с которой начинаются почти все Win32-программы. Таким образом, если каким-нибудь способом удастся узнать, с какого адреса происходит вызов GetModuieHandie, оригинальная точка входа с большой вероятностью будет где-то неподалеку.

Есть и другой способ выявления оригинальной точки входа. Он заключается в том, чтобы в момент, когда протектор уже расшифровал секцию кода, но еще не передал в нее управление, с помощью функции WriteProcessMemory заполнить всю секцию кода байтами со значением OxCC (что соответствует команде процессора Int3 — вызов отладочного прерывания). Разумеется, такой код не может выполняться, о чем операционная система и известит пользователя. А в некоторых случаях (в частности, под Windows NT, 2000 и XP) еще и сообщит адрес, по которому произошла ошибка. Этот адрес и будет являться адресом оригинальной точки входа. Данный метод известен со времен DOS, где он использовался, в частности, для поиска оригинальных точек входа в прерывания.

Кстати, во времена DOS также существовали упаковщики и протекторы исполняемых файлов, и для противодействия им разрабатывались автоматические депротекторы. Фактически, труднее всего автоматизации поддавалась именно задача поиска оригинальной точки входа. И существовал депротектор, носивший название Intruder (вторгающийся), который умел в процессе запуска программы определять, каким компилятором она была создана, и, исходя из этого, вычислял правильный адрес точки входа. Intruder "знал" практически все распространенные в то время средства разработки и в по-

давляющем большинстве случаев успешно снимал навесную защиту в автоматическом режиме.

С извлечением ресурсов больших проблем обычно не возникает. Протектору приходится расшифровывать и настраивать ресурсы в памяти таким образом, чтобы функции Win32 API, отвечающие за доступ к ресурсам, могли нормально работать. Следовательно, действуя точно так же, как действуют функции Win32 API, из загруженной в память программы можно извлечь каждый ресурс в отдельности, и тогда для восстановления секции ресурсов останется только построить дерево ресурсов (в соответствии со спецификацией формата Portable Executable).

Данные программы могут изменяться в процессе выполнения. Следовательно, для получения неискаженных секций данных необходимо читать их из памяти процесса именно в тот момент, когда протектор передает управление основной программе. Если оригинальная точка входа известна, определение такого момента не составит большого труда. Для этого можно использовать отладочные регистры центрального процессора.

Начиная с Intel 80386 все процессоры семейства x86 имеют аппаратные средства для отладки приложений. Процессор позволяет использовать до 4-х аппаратных точек останова. Каждая точка описывается типом доступа, который будет отслеживаться процессором (чтение, запись, выполнение), адресом, при обращении по которому процессор сгенерирует исключение, и размером контролируемой области (BYTE, WORD, DWORD). Для работы с аппаратными точками останова используются так называемые отладочные регистры, которые в системе команд x86 имеют логические имена, начинающиеся с DR (Debugging Registers). Достаточно установить точку останова на исполнение кода по адресу, соответствующему найденной оригинальной точке входа, обработать генерируемую процессором исключительную ситуацию и прочитать содержимое памяти.

Регистры аппаратной отладки не доступны напрямую пользовательским программам, т. к. операции чтения и записи отладочных регистров являются привилегированными и разрешены только в режиме ядра (в драйверах). Но к содержимому этих регистров можно получить доступ несколькими способами и без драйвера.

Во-первых, содержимое всех регистров в каком-то потоке может быть получено и установлено через такие функции Win32 API, как `GetThreadContext` и `SetThreadContext`. Перед обращением к контексту потока рекомендуется остановить поток функцией `SuspendThread`, а по окончании манипуляций с контекстом запустить его вновь при помощи `ResumeThread`.

Во-вторых, существует подмножество Win32 API, называемое Debugging API (программный интерфейс отладки). С помощью функций, входящих в со-

став Debugging API, очень просто написать собственный отладчик, который будет получать извещения обо всех важных событиях и исключениях, возникающих в отлаживаемой программе. И отладчик также может использовать функции `GetThreadContext` и `SetThreadContext` для доступа к отладочным регистрам.

И наконец, в-третьих, изнутри программы доступ к отладочным регистрам может быть осуществлен путем использования механизма структурированной обработки исключений — Structured Exception Handling (SEH). Достаточно установить свой обработчик исключения и выполнить заведомо неправильную операцию (например обращение по адресу 0x00000000). При возникновении ошибки будет вызван обработчик исключения, которому операционная система передаст указатель на структуру контекста потока, содержащую значение всех регистров. При этом значения, которые окажутся в этой структуре после завершения обработчика исключения, будут занесены в соответствующие регистры центрального процессора, включая отладочные.

Самую значительную трудность при снятии защиты, наверное, представляет собой восстановление таблиц импорта. Однако и эта задача может быть успешно решена.

Следует отметить, что большинство протекторов оставляют в защищенной программе таблицу импорта, содержащую как минимум по одной импортируемой функции из каждой библиотеки, использованной в оригинальной программе, иначе загрузчик, являющийся частью операционной системы, вообще не отобразит библиотеку в адресное пространство процесса. Это позволяет легко определить точный список библиотек, из которых импортируются функции. В крайнем случае, если перечень импортируемых библиотек получить не удалось, существует возможность узнать, какие вообще библиотеки были подключены к процессу. Это будут все библиотеки, на которые есть ссылки в таблицах импорта незащищенной программы, и некоторое количество других библиотек.

После того как все используемые динамические библиотеки оказались отображены в адресное пространство программы, можно получить адрес каждой экспортируемой функции для всех необходимых библиотек.

Теперь остается только найти в оперативной памяти, принадлежащей программе, расположение таблицы адресов импортированных функций и определить размер этой таблицы.

После загрузки и настройки программы в памяти таблица адресов импортируемых функций имеет обычно следующий вид. Сначала идут несколько 32-битовых значений, являющихся адресами функций, импортируемых из первой библиотеки. Последовательность указателей, относящаяся к первой

даблиотеке, заканчивается нулевым элементом. Сразу за ним идут указатели ıa импортируемые функции второй библиотеки, которые снова заканчива-ɔтся нулем, и так далее для всех библиотек.

'аким образом, следует искать последовательности 32-битовых значений, аканчивающиеся нулевым элементом, и все элементы последовательности ɩолжны совпадать с адресами функций одной из библиотек. Каждая такая ɩоследовательность будет относиться к одной из библиотек, а все последо-;ательности вместе будут образовывать таблицу адресов импортируемых пункций.

'едактор связей (linker) обычно размещает таблицу адресов импортируемых ɔункций в секции статических данных, так что имеет смысл начинать поиск [менно оттуда.

Тосле того как определены все функции, необходимо построить и осталь-[ые таблицы, отвечающие за импорт, но это делается в соответствии со пецификацией формата переносимого исполняемого файла и очень легко втоматизируется.

'ешение самой последней задачи — удаление кода протектора — на самом еле не является необходимым для получения работоспособной программы. :сли восстановить зашифрованные секции, индексы ресурсов, таблицы им-орта, найти точку входа и создать новый исполняемый файл, в котором удет присутствовать код протектора, он должен работать и так. Но, удалив од и данные, добавленные протектором, можно несколько сократить раз-ıер исполняемого файла.

3.4. Борьба технологий ıащиты и взлома

азумеется, разработчики протекторов не хотят мириться с тем, что восста-овить оригинальную программу так легко. И они всячески стремятся если е сделать восстановление невозможным, то хотя бы максимально услож-ить этот процесс и, самое главное, предотвратить полную автоматизацию. ,ело в том, что для использования автоматического депротектора не нужно ыть гением, достаточно иметь базовые знания. А вот деактивировать дейст-ительно серьезную навесную защиту могут буквально единицы — считан-ые проценты от числа всех людей, серьезно занимающихся исследованием рограмм (Reverse Engineering), и тысячные доли процента от общего числа ользователей.

днако специалисты по исследованию программ тоже не хотят мириться тем, что протектор не удается быстро снять, и придумывают новые методы

обхода защиты. Такое неофициальное противостояние продолжается не первый год, и конца ему пока не видно. Но, похоже, именно это противостояние и стимулирует развитие протекторов. Иначе уровень технологических решений, используемых для защиты, остался бы на уровне, достигнутом еще в первых версиях протекторов.

Для того чтобы код (исполняемые инструкции) нельзя было прочитать из памяти после запуска программы, применяется шифрование отдельных участков кода с расшифровкой их непосредственно перед началом и зашифровкой сразу по окончании выполнения.

Одним из способов достижения такого поведения программы является использование API протектора. То есть программист, разрабатывающий программу, которая позже будет подвергнута обработке протектором, специальным образом описывает, какие области должны быть зашифрованы большую часть времени выполнения программы, и явным образом указывает, в какой момент должны производиться расшифрование фрагмента и его последующее зашифрование, обращаясь к API протектора.

Обычно эта схема применяется в сочетании с использованием регистрационных кодов или лицензионных файлов. Пока пользователь имеет в своем распоряжении только незарегистрированную (ограниченную) версию программы, некоторые функции хранятся в зашифрованном виде и не могут быть выполнены. Если же пользователь приобретает лицензию на программу и вводит правильный регистрационный код (или указывает расположение лицензионного файла), программа получает возможность расшифровать и выполнить зашифрованные фрагменты, обеспечив тем самым полную функциональность.

Другой способ защиты кода заключается в использовании особенностей работы процессора. Можно модифицировать некоторые фрагменты программы таким образом, чтобы при передаче им управления возникали так называемые исключительные ситуации (Exception). Эти исключительные ситуации будут обработаны кодом протектора, который должен восстановить правильное содержимое модифицированного фрагмента кода, позволить ему выполниться, а затем снова привести к неработоспособному состоянию.

Для сокрытия адреса точки входа можно, например, первые несколько десятков или сотен байт скопировать внутрь тела протектора и передать управление оригинальному коду программы уже после того, как часть команд была выполнена. В результате, некоторое количество команд, выполняемых только один раз при старте программы, просто будут отсутствовать в коде программы.

Для программ, написанных с использованием "чистого" Win32 API (без библиотек типа Object Windows Library или Visual Components Library), ресурс

окна часто подгружается в память неявно путем вызова функции CreateDialogParam ИЛИ CreateDialogIndirectParam. При ЭТОМ не сама программа, а менеджер диалогов, являющийся частью операционной системы, читает ресурс, описывающий окно, и интерпретирует его, подгружая из ресурсов меню, картинки и т. д.

Однако некоторые средства разработки сохраняют описания диалогов в собственном формате и практически не полагаются на то, как работает с ресурсами менеджер диалогов, встроенный в Windows. Это характерно, например, для программ, созданных при помощи Borland Delphi или C++ Builder с использованием библиотеки Visual Components Library' (VCL).

Ресурс, описывающий диалоговое окно в VCL, хранится как двоичный блок данных RCDATA, и Windows не пытается самостоятельно его читать и интерпретировать. Поэтому протектор может модифицировать программу таким образом, чтобы ресурсы, описывающие диалоги, сохранялись в зашифрованном виде и расшифровывались только в тот момент, когда программа обращается к VCL с запросом на загрузку диалога из ресурсов. Этот метод позволяет лучше защитить ресурсы, но, разумеется, применим далеко не ко всем программам.

С защитой данных можно поступать так же, как и с ресурсами, если известны особенности использованного компилятора и редактора связей. Очень многие программы, созданные в определенных средах разработки, начинают выполнение с одних и тех же действий. Протектор может определить, если защищаемая программа начинается именно с таких действий, и перенести их выполнение в тело протектора. Таким образом, к тому моменту как управление будет передано защищенной программе, протектор уже выполнит часть настроек, но позаботится о том, чтобы программа их не выполняла повторно. Следовательно, в оригинальной точке входа программа содержит модифицированные данные, и получить на их основе работоспособную копию со снятой защитой будет проблематично.

Для защиты содержимого таблицы адресов импортируемых функций могут применяться весьма разнообразные подходы. Например, протектор может заполнить таблицу импорта ссылками на маленькие функции-переходники, расположенные в теле протектора. А каждый переходник будет передавать Управление нужной функции во внешней библиотеке. Как расширение этого метода, внутрь функции-переходника из библиотечной функции может копироваться несколько первых инструкций, которые будут выполняться в теле протектора. Следовательно, управление будет передаваться не на Первую инструкцию библиотечной функции. Это не только затрудняет восстановление таблиц импорта, но и позволяет обойти точки останова, размещенные в начале библиотечных функций.

Некоторые функции Win32 API во время выполнения процесса всегда возвращают одно и то же значение. Примером такой функции является GetCommandLine. Протектор может вызвать эту функцию до передачи управления основной программе и запомнить результат, а на запросы программы к GetcommandLine просто возвращать сохраненное значение. При этом передачи управления библиотечной функции вообще не происходит.

Наконец, в распоряжении разработчиков протектора есть довольно сложное в реализации, но очень эффективное средство — трансляция части инструкций в псевдокод и выполнение этого псевдокода на встроенной виртуальной машине. То есть к моменту передачи управления в тело защищенной программы некоторые фрагменты кода, относящиеся или к самой программе, или к импортированным функциям, представлены не в системе команд процессора семейства x86, а в некотором альтернативном виде, и выполнить эти фрагменты кода может только протектор. Следовательно, для снятия защиты необходимо разобраться в системе команд виртуальной машины и перевести защищенные фрагменты из системы команд протектора в систему команд процессора x86, а это весьма трудоемкая задача.

Как уже упоминалось ранее, некоторые средства защиты имеют свой собственный API, позволяющий защищаемой программе обращаться к протектору во время выполнения. Это способствует интеграции протектора с программой и создает очевидные трудности, связанные с необходимостью эмуляции API протектора при снятии защиты.

Также разработчики некоторых протекторов предлагают тем, кто хочет защищать свои программы, различные способы, помогающие установить факт, что с программы была снята защита. Эти способы могут являться частью API протектора, а могут и основываться на характерных признаках, которыми обладает защищенная программа после ее загрузки в память.

Очень многие протекторы усиленно препятствуют использованию средств отладки, включая отладочные регистры. Одним из методов, мешающих использованию аппаратных точек останова, является хранение промежуточных данных протектора в отладочных регистрах, следствием чего является отказ в запуске программы, если содержимое отладочных регистров меняется извне.

Но, как уже было сказано выше, исследователи программ постоянно совершенствуют методы атаки на протекторы, и если такие способы защиты, как виртуальная машина, способны противостоять автоматическим депротекторам, то против ручной распаковки, наверное, не способен устоять ни один из существующих протекторов.

13.5. Несколько интересных протекторов

В настоящий момент разработано достаточно большое число протекторов исполняемых файлов. Многие из них бесплатны и созданы энтузиастами, которым просто интересно попробовать свои силы в защите программ. Разумеется, есть и коммерческие протекторы. А некоторые протекторы являются составляющими частями более сложных комплексов, включающих в себя привязку к аппаратным ключам или компакт-дискам.

Рассмотрим основные характеристики нескольких наиболее интересных протекторов.

13.5.1. ASProtect

Для защиты условно бесплатных программ чаще всего, наверное, применяется ASProtect — протектор, разработанный Алексеем Солодовниковым. ASProtect был чуть ли не первым серьезным протектором, сочетавшим в себе основные функции, применяемые для защиты программ:

- работа с регистрационными кодами на базе RSA-1024;
- поддержка "черного списка" регистрационных кодов;
- ограничение периода работы пробной версии;
- ограничение функциональности пробной версии;
- динамическое расшифрование фрагментов кода при наличии правильного регистрационного кода;
- API для интеграции защищаемой программы с протектором;
- оригинальные методы противодействия исследованию под отладчиком;
- оригинальные методы защиты от снятия протектора.

Однако благодаря огромной популярности ASProtect является и одним из самых хорошо изученных протекторов — почти для всех хитростей, применяемых в ASProtect, разработаны или автоматические, или полуавтоматические средства обхода.

Иногда у программ, защищенных ASProtect, возникают проблемы с работой под новыми версиями операционных систем, но автор не прекращает работы по совершенствованию протектора и стремится оперативно исправлять все обнаруженные ошибки, а также добавлять новые защитные механизмы.

13.5.2. Armadillo

Непривычный метод взаимодействия с защищаемой программой использует протектор, разработанный компанией The Silicon Realms Toolworks и нося-

щий название Armadillo. При запуске защищенная программа выполняется как 2 процесса. Первый процесс, в котором работает основной код протектора, создает в режиме отладки второй процесс, содержащий собственно защищенную программу, и управляет его выполнением.

Протектор Armadillo применяет оригинальные технологии, называемые CopyMemII и Nanomites, для защиты кода выполняемой программы от считывания из памяти. Технология CopyMemll уже хорошо изучена и легко обходится автоматическими депротекторами. Про существование автомата, способного обойти Nanomites, пока не известно, но были многочисленные сообщения о ручной распаковке программ, при защите которых использовалась эта технология.

Протектор Armadillo также включает в себя менеджер лицензий.

13.5.3. PACE InterLok

Протектор InterLok, разработанный компанией PACE Anti-Piracy, имеет версии для Windows и Macintosh. Набор функций, предлагаемых протектором, вполне обычный: менеджер лицензий, демонстрационные версии, API для интеграции и т. д.

Версия для Windows устанавливает драйвер ядра, препятствующий отладке защищенного приложения и выполняющий часть проверок. Но, несмотря на наличие такого сложного элемента, как драйвер, защита исполняемого файла довольно слабая. Например, все секции шифруются потоковым шифром с одним и тем же ключом. Это приводит к тому, что если секция кода больше, чем любая другая секция, то можно прочитать из памяти запущенной программы расшифрованную секцию кода, вычислить гамму, накладываемую при шифровании, и расшифровать все остальные секции файла, даже не прибегая к сложным инструментам. Драйвер практически не имеет защиты от исследования, а таблицы импорта вообще никак не защищены.

13.5.4. HASP Envelope

В комплект разработчика, поставляемый вместе с ключами HASP, входит протектор HASP Envelope. Цель этого протектора — защитить программу от исследования и даже запуска при отсутствии аппаратного ключа HASP.

Так как секретная функция, являвшаяся на протяжении многих лет сердцем ключей HASP, оказалась полностью раскрыта, не составляло большого труда эмулировать ответы на запросы, которые Envelope делал к ключу. Следовательно, любая программа, защищенная HASP Envelope, могла быть запущена без ключа.

С появлением ключей семейства HASP4, в которых используются новые секретные функции `HaspEncodeData` и `HaspDecodeData`, обход Envelope при отсутствии ключа стал невозможен. Но в остальном протектор не способен обеспечить высокую стойкость защиты. И при наличии ключа получение незащищенной копии программы обычно не требует значительных усилий.

13.5.5. StarForce

Одной из составляющих StarForce Professional (системы, разработанной компанией Protection Technology для защиты информации, распространяемой на компакт-дисках) является протектор. В его функции входят проверка подлинности компакт-диска и запуск защищенной программы, но только в том случае, если введенный пользователем лицензионный код соответствует установленному в приводе диску.

В StarForce применяется множество уникальных технологических решений. Так, например, в защищенном исполняемом файле секция кода заполнена одними нулями, а код защиты выполняется из динамической библиотеки protect.dll, которая автоматически загружается и инициализируется при запуске программы.

Большая часть защиты сосредоточена в драйвере, устанавливаемом в ядро операционной системы. Именно там идет проверка подлинности компакт-диска. Сам драйвер также зашифрован с целью затруднения его исследования.

Часть защищаемой программы хранится в псевдокоде и выполняется на встроенной в протектор виртуальной машине.

Несмотря на то, что в Интернете можно найти подробные описания процесса ручного снятия StarForce с некоторых программ, таких случаев крайне мало. Частично это может быть объяснено тем, что для исследования защиты необходимо наличие оригинального компакт-диска, а также тем, что основная категория продуктов, которые защищаются с помощью StarForce, — компьютерные игры. А взлом защиты игр экономически не очень выгоден, т. к. стоимость одной копии игры невелика, а период популярности весьма короток. Но надо отдать должное разработчикам — они неплохо потрудились.

В целом StarForce на сегодняшний день является одним из самых серьезных протекторов, который, к тому же, продолжает развиваться. Так недавно компания Protection Technology объявила о выходе StarForce Soft 3.0 — системы защиты от копирования, основанной на ядре StarForce Professional 3.0, но не использующей компакт-дисков.

13.6. Что плохого в протекторах

Протекторы призваны обеспечить защиту содержимого исполняемого файла от исследования и модификации. Но часто получается, что защищенная программа по некоторым характеристикам оказывается для пользователя хуже, чем та же программа, но без защиты.

13.6.1. Расход памяти

Прежде всего, протектор — это дополнительный объем кода. Если протектор не поддерживает упаковку защищаемых данных, размер файла на диске в процессе защиты увеличится на размер самого протектора. Правда, стоит отметить, что при использовании сжатия защищенный файл может оказаться меньше оригинального в несколько раз.

Также код протектора требует и некоторого количества оперативной памяти. Но основной перерасход ресурсов оперативной памяти происходит по другой причине, связанной с особенностью загрузки исполняемых файлов в Win32.

Практически все современные операционные системы имеют встроенную поддержку так называемых файлов страничной подкачки (Page File или Swap File). Вся логическая память, доступная выполняемым программам, разбивается на страницы, и некоторые редко используемые страницы могут оказаться не в оперативной памяти, а на диске в файле подкачки. При любом обращении к такой странице менеджер памяти выполняет операцию чтения данных с диска в оперативную память (подкачку). Если в оперативной памяти нет свободных страниц, одна из наиболее редко используемых страниц вытесняется из оперативной памяти на диск. В Win32 также существует механизм файлов, отображаемых в память (Memory Mapped Files), который позволяет отобразить в адресное пространство любой дисковый файл.

Для ускорения загрузки и выполнения программ как раз и используется отображение файлов в память. Вместо того чтобы читать с диска весь файл программы, необходимые области просто отображаются в оперативную память нового процесса, а реальный перенос данных с диска производится непосредственно в тот момент, когда происходит обращение к каждой конкретной странице.

Более того, если загружается несколько копий одной программы, то те страницы памяти, которые не меняются в процессе работы, размещаются в оперативной памяти только один раз, независимо от числа процессов, которые эту память используют. Но как только один из процессов изменяет

содержимое страницы, для него создается персональная копия измененной страницы, под которую выделяется дополнительная память, а остальные процессы продолжают совместно использовать оригинальную страницу.

Как уже говорилось раньше, в обычных программах существуют области, не изменяемые в процессе работы программы. Это, например, секции кода и ресурсов. Таким образом, эти области будут подгружаться с диска по мере использования, и если запустить несколько копий программы, то оперативная память под неизменяемые области будет израсходована только один раз.

Если же программа обрабатывается протектором, то это влечет за собой определенные последствия. Во-первых, замедляется запуск программы, т. к. протектор вынужден прочитать с диска весь код программы до начала выполнения, чтобы правильно настроить программу в памяти. Для небольших программ это, возможно, будет почти незаметно, но с увеличением размера файла задержки при старте защищенной программы могут стать весьма ощутимыми. А во-вторых, при расшифровке тела программы протектор вынужден модифицировать содержимое страниц памяти, а значит, для каждого процесса будет создаваться локальная копия. И если программа должна присутствовать в памяти в нескольких экземплярах, то области кода и ресурсов будут занимать во столько раз больше памяти, сколько процессов запущено.

13.6.2. Безопасность

Разработчики протекторов стремятся использовать все доступные им возможности для повышения стойкости. Однако иногда они забывают, что, повышая защиту конкретной программы, они могут случайно ослабить некоторые элементы защиты.

Драйвер ядра в StarForce

Для обеспечения функционирования некоторых элементов защиты и затруднения анализа защитных механизмов разработчики системы защиты StarForce создали специальный драйвер, который устанавливается вместе с защищаемой программой и без которого функционирование этой программы невозможно.

Станислав Винокуров, являющийся сотрудником компании SmartLine, Inc., обнаружил, что некоторые версии драйвера StarForce содержат ошибку, позволяющую программе сформировать правильные структуры данных и, воспользовавшись установленным драйвером, выполнить любую последовательность команд в режиме ядра.

Фактически, наличие данной ошибки позволяет получить неограниченный доступ к ресурсам компьютера, на котором установлен драйвер StarForce, полно-

стью нейтрализовав защитные механизмы операционных систем семейства Windows NT.

Правда по утверждению представителя компании Protection Technology, являющейся разработчиком StarForce, в последних версиях драйвера эта ошибка уже устранена.

13.6.3. Нестабильность

Для того чтобы усложнить работу исследователя программ, разработчики протектора вынуждены идти на использование нестандартных и/или недокументированных особенностей операционных систем и оборудования. Как правило, эти особенности обнаруживаются путем исследования внутренностей самой операционной системы, а их наличие подтверждается путем многократного тестирования.

Но разнообразие версий программного обеспечения и оборудования настолько велико, что небольшой компании просто физически не удастся проверить обнаруженную особенность на всех конфигурациях, которые могут встретиться у пользователя. Но и крупным корпорациям это далеко не всегда под силу.

Кроме того, с некоторой периодичностью появляются новые версии операционных систем. И очевидно, что любая недокументированная особенность, существовавшая ранее, может отсутствовать в новой версии операционной системы.

Все это приводит к тому, что защищенная программа с некоторой вероятностью может просто не заработать на машине у конкретного пользователя (или у всех пользователей, использующих новую операционную систему или установивших у себя какую-то особенную программу).

Иногда разработчики протекторов излишне усердствуют, стараясь воспрепятствовать анализу защищенных программ. Как уже упоминалось ранее, многие протекторы запрещают выполнять программу под отладчиком. Но не всегда делают это достаточно разумными средствами.

Нелепая защита от отладчика

При запуске, например, исполняемых файлов, входящих в состав программного продукта PHOTOMOD, защищенного с помощью HASP Envelope, нередко можно получить сообщение об ошибке, гласящее, что программа не будет работать, т. к. в памяти обнаружен отладчик. И подобное сообщение, скорее всего, немало удивит пользователя, совершенно уверенного в том, что он не запускал никакого отладчика.

Оказывается, защищенная программа так реагирует, в частности, на наличие в памяти процесса с именем MSDEV.EXE. Для справки, MSDEV.EXE — это имя основного файла Microsoft Developers Studio — набора средств разработки программного обеспечения, выпускаемого корпорацией Microsoft. Действительно, MSDEV.EXE может использоваться и для отладки программ, но основное его предназначение — именно разработка. То есть авторы PHOTOMOD считают, что пользователи их продукта ни в коем случае не должны пользоваться средствами разработки, предоставляемыми компанией Microsoft.

Самое забавное в этом примере то, что после переименования MSDEV.EXE его можно держать загруженным в памяти и это не повлияет на работоспособность PHOTOMOD. Следовательно, такое обнаружение отладчика скорее мешает работать обычному пользователю, чем ставит хоть сколько-нибудь ощутимое препятствие на пути исследователя.

Другой пример касается нескольких версий программы Adobe Acrobat eBook Reader, защищенной при помощи протектора PACE InterLok. Защита использовала драйвер, устанавливаемый в ядро операционной системы, и одной из функций драйвера была борьба с отладчиком.

При запуске программа обращалась к драйверу и извещала его о том, что выполнение критической части (требующей блокировки работы отладчика) началось. Перед завершением работы программы драйвер извещался о том, что критическая часть пройдена и блокировку можно отключить. Собственно блокировка заключалась в том, что при возникновении одного из отладочных исключений (например пошагового выполнения или точки останова) драйвер без лишних вопросов выполнял перезагрузку компьютера.

Если бы драйвер так реагировал только на отладочные исключения, возникающие в контексте защищаемой программы, это еще можно было бы понять. Но перезагрузка выполнялась при исключении в любом процессе. То есть попытка отладки программы в том же Microsoft Developers Studio при загруженном Adobe Acrobat eBook Reader почти неминуемо приводила к перезагрузке с потерей всей несохраненной информации.

В последних версиях Adobe Acrobat eBook Reader проблема, похоже, была решена путем объявления критическими только самых важных частей программы, а не всей программы целиком.

Грамотное использование протекторов позволяет значительно усложнить задачу снятия защиты и если не предотвратить взлом, то значительно снизить его рентабельность. Нелегальные копии программ с хорошей зашитой часто появляются в Интернете только благодаря кардингу — если взломать программу не удается, можно купить лицензию, воспользовавшись украденным номером кредитной карты.

Глз

Пр
pal

В про
стоит
ва, что
содер)
не вос

14.1

Хорош
дачи н
подхо
состав
незави

Для т
танны
ствия,
легко
a Devi

При к
сены
ни фу
рые э
(напр
файло
возни
кнопк
Дочно
этих с
адресу

ıва 14



иемы, облегчающие ɔоту противника

ıцессе разработки реализации защитных механизмов ни на минуту не забывать, что противник будет использовать все доступные ему средст- ɔбы нейтрализовать защиту. Однако очень часто защищенная программа кит в себе такое количество информации, способствующей взлому, что :пользоваться ею для обхода защиты было бы для противника странно.

I. Осмысленные имена функций

ıим тоном при разработке сложных проектов является разделение за- ма отдельные, минимально связанные между собой части. При таком ɖе жестко определяются интерфейсы, с помощью которых отдельные ıлющие могут общаться между собой, и каждая часть разрабатывается ıсимо от всех остальных.

ого чтобы программистам было проще обращаться к узлам, разрабо- ıМ другими людьми, функциям, через которые происходят взаимодей- , дают легко запоминающиеся и самоописывающие имена. Например, запОМНИТЬ, ЧТО `CreateFile` используется ДЛЯ создания файла, `.ceIoControl` — для управления устройствами.

compиляции программы некоторые фрагменты кода могут быть выне- во внешние библиотеки и импортированы по именам. Иногда по име- 'нкции можно восстановить даже количество и тип аргументов, кото- га функция принимает на вход. Многие визуальные среды разработки имер Borland Delphi и C++ Builder) сохраняют внутри исполняемых ɔв строки с именами функций, являющихся обработчиками событий, кающих в интерфейсе программы (перемещение мыши, нажатие :и и т. д.). А иногда в готовой программе остаются фрагменты отла- й информации, в которой присутствуют имена функций. В любом из :лучаев не составляет большого труда определить, что по конкретному ' начинается код функции с таким-то именем.

Это не является проблемой для функций, выполняющих действия, не связанные с защитой программы. Но если функция называется CheckLicense или btnRegisterClick, то противнику имеет смысл в первую очередь исследовать именно такую функцию, т. к. с большой вероятностью в ней выполняются важные действия, относящиеся к работе защиты.

Стоит всячески избегать попадания осмысленных имен функций, относящихся к защитным механизмам, в исполняемые файлы и библиотеки, являющиеся частью готовой программы. Если в исходном тексте программы функции должны фигурировать под осмысленными именами (по соображениям удобочитаемости), то для сокрытия истинных имен функций можно использовать макроподстановки (листинг 14.1).

Листинг 14.1. **Использование #define** для сокрытия **имен функций в** С

```
#define CheckLicense fn23

void CheckLicense (char *pszLic) {
  /* текст функции */


void main (void) {
  CheckLicense ("License Sting");
```

При компиляции такого кода препроцессор вместо CheckLicense везде подставит fn23, а значит, имя, которое могло бы дать какую-то информацию противнику, просто не попадется ему на глаза.

Библиотечные функции могут импортироваться и экспортироваться не только по имени (by name), но и по номеру (by ordinal). Это позволяет вообще исключить соответствующие имена из программы и библиотек.

Однако никогда не помешает в готовых исполняемых файлах выполнить поиск текстовых строк, содержащих названия важных для защиты функций — никогда нельзя быть уверенным, что компилятор или редактор связей нигде не оставил важных имен.

14.2. Транслируемые языки

Некоторые широко распространенные языки программирования в процессе компиляции преобразуют исходный текст в так называемый псевдокод —

некоторое промежуточное представление текста программы, не являющееся машинным кодом. К таким языкам можно отнести Clipper, C#, FoxPro, инсталляционные сценарии InstallShield, Java, MapInfo Map Basic, MicroStation MDL, Python, Visual Basic и многие другие. При выполнении программы виртуальная машина интерпретирует псевдокод и выполняет его на виртуальном процессоре.

Теоретически использование виртуальной машины может являться эффективным способом противодействия исследованию программы, т. к. до начала анализа алгоритма необходимо разобраться с устройством виртуальной машины. Но это справедливо только для ситуации, когда система команд, применяемая машиной, нигде и никем не была описана, т. е. является уникальной.

Очевидно, что для популярных языков программирования это совершенно не так. Для некоторых языков (например C#, Java и Python) в Интернете нетрудно найти подробное описание того, как кодируется та или иная операция, поддерживаемая виртуальной машиной. А интерпретатор языка Python вообще распространяется в исходных текстах, что не позволяет сохранять устройство виртуальной машины в тайне.

Если для какого-то языка нет описания кодов операций, но этим языком пользуется довольно много программистов, рано или поздно кто-то задастся целью разобраться в деталях псевдокода и разработает весь необходимый инструментарий.

Существует несколько причин, почему разобраться с системой команд виртуальной машины для транслируемого языка программирования обычно бывает не очень сложно.

Прежде всего, исследователь может компилировать любые примеры и смотреть, в какой псевдокод будут превращаться команды. Это очень важно, т. к. внося незначительные изменения в исходный текст и анализирую разницу в оттранслированном псевдокоде, гораздо легче устанавливать закономерности, чем внося изменения в псевдокод и контролируя изменения в поведении виртуальной машины.

Кроме того, виртуальная машина обычно проектируется, исходя из требований максимизации производительности. Следовательно, знание базовых принципов построения эффективных виртуальных машин часто позволяет быстро разобраться с особенностями конкретной реализации.

Вдобавок система команд виртуальной машины редко бывает очень сложной. Это на реальном процессоре обнулить регистр eax можно несколькими способами, например:

sub eax, eax; xor eax, eax; and eax, eax; mov eax, 0.

А в виртуальной машине такая избыточность не имеет смысла.

И, наконец, виртуальная машина, являющаяся частью языка программирования, не разрабатывается как запутанное логическое устройство, работа которого никогда не должна быть проанализирована противником. Напротив, чем проще будет организована виртуальная машина, тем легче будет ее отлаживать и оптимизировать.

В любом случае, для многих популярных транслируемых языков программирования были разработаны декомпиляторы, позволяющие получить если не точную копию оригинального исходного текста, то часто эквивалентный код, который может быть скомпилирован и будет работать так же, как оригинальная программа. В разных языках программирования количество информации, сохраняемой в оттранслированном тексте, может отличаться. В некоторых случаях сохраняются имена всех функций и переменных. Иногда могут быть извлечены даже номера строк исходного текста, где располагался тот или иной оператор, и имя исходного файла. Но может быть и так, что сохраняется только последовательность вызовов функций и употребления операторов, в то время как вся символическая информация об именах оказывается утраченной.

Но, даже имея всего лишь эквивалентный текст, в котором имена переменных и функций заменены на произвольные, анализировать защитные механизмы гораздо проще, чем если бы они были написаны на языке, компилируемом в команды реального процессора.

Для предотвращения применения декомпиляторов иногда разработчики программ идут на модификацию виртуальной машины.

Модификация виртуальной машины FoxPro

FoxPro является одной из популярных коммерческих систем управления базами данных (СУБД) и позволяет создавать законченные приложения, функционирующие независимо от среды разработки. Но существует несколько очень хороших декомпиляторов, способных полностью восстановить исходный текст программ, созданных в FoxPro, например ReFox или UnFoxAll. И некоторые разработчики, использующие FoxPro (например авторы программы Hardware Inspector), пытаются найти способ защитить свои программы от декомпиляции. Делается это примерно следующим образом.

Программа разрабатывается в среде FoxPro и компилируется самым обычным образом. Чтобы ReFox невозможно было использовать для получения исходного текста программы, необходимо изменить способ кодирования псевдокода. Но тогда и виртуальная машина не сможет работать с перекодированным псевдокодом. Следовательно, необходимо исправить и виртуальную машину.

После этого скомпилированную программу можно распространять вместе с модифицированной виртуальной машиной, и ReFox окажется бессилен.

Однако в данной схеме есть одно слабое звено. Дело в том, что исполняющая часть виртуальной машины обычно оформляется в виде динамической библиотеки (vfp500.dll для FoxPro 5 или vfp6r.dll для FoxPro 6) и разрешается свободное распространение этой библиотеки (как redistributable component). Следовательно, оригинальная (неизмененная) версия виртуальной машины может быть легко найдена в Интернете. Далее достаточно выяснить, чем отличается модифицированная версия виртуальной машины, и либо перекодировать программу, приведя ее к виду, доступному для понимания ReFox, либо модифицировать ReFox таким же образом, каким была модифицирована виртуальная машина.

Так что модификация виртуальной машины является весьма сомнительным средством для защиты от декомпиляции псевдокода. К тому же, распространение модифицированной виртуальной машины почти всегда является нарушением условий лицензии, в согласии с которыми эта машина должна использоваться. В случае с виртуальной машиной FoxPro, являющейся собственностью корпорации Microsoft, ущемленными оказываются права последней, а это может вызвать серьезные последствия.

Одним словом, использование транслируемых языков программирования, допускающих частичную или полную декомпиляцию, — это далеко не лучший выбор для реализации защитных механизмов. Противник имеет возможность в самые короткие сроки получить доступ к исходным текстам всех исходных алгоритмов, чтобы попытаться найти уязвимость. А иногда противнику и вовсе удается убрать из декомпилированного текста все обращения к защитным функциям и заново скомпилировать программу.

14.3. Условно бесплатные и Demo-версии

Для того чтобы потенциальные пользователи смогли лучше оценить возможности программы, разработчики часто распространяют демонстрационные версии своих продуктов. Такие версии, как правило, имеют ограниченный набор функций и/или ограничение на время использования или количество запусков программы.

Условно бесплатные продукты обычно являются ограниченными версиями, которые предоставляют пользователю возможность ввода регистрационного кода, после чего все ограничения снимаются.

14.3.1. Ограничение функциональности

Если автор демонстрационной версии программы хочет сделать недоступным, например, пункт меню Save, то он может пойти двумя путями:

- при инициализации программы сделать этот пункт недоступным;
- удалить из программы весь код, относящийся к сохранению данных на диске, и при инициализации программы сделать пункт меню Save недоступным.

Очевидно, что реализация первого способа требует значительно меньших усилий. Однако существуют специальные инструменты, позволяющие менять свойства элементов диалога в процессе выполнения программы. С помощью подобных инструментов можно каждый элемент любого диалогового окна сделать доступным, превратив демонстрационную версию в полноценную по функциональности программу. А некоторые инструменты можно даже "обучить" автоматически делать доступными нужные кнопки и пункты меню при открытии соответствующего диалога.

Поэтому необходимо исключить из кода демонстрационной программы те функции, которые должны присутствовать только в полной версии. Достичь желаемого результата, не создавая две очень похожих программы, можно, например, путем использования директив условной компиляции, поддерживаемых препроцессором языка С. К полезным директивам относятся, например, #define, #ifndef, #ifdef, #else и #endif. С их помощью можно добиться того, что, изменяя в настройках проекта всего одно определение препроцессора (аналог #define), можно будет из одного набора исходных текстов получить совершенно разные по набору функций программы.

14.3.2. Ограничение периода использования

Для того чтобы ограничить период возможного использования продукта, необходимо в некоторой области компьютера сохранить дату установки или количество запусков. Обычно для этого используются произвольные файлы или реестр (Registry Database). Многие считают, что, спрятав такой счетчик в самый дальний угол операционной системы, они сделают невозможным обнаружение его местоположения, а следовательно, и сброс счетчика.

Однако существует два семейства инструментов, позволяющих определить, где именно располагается счетчик. К первому семейству относятся программы-мониторы. Они отслеживают все обращения к файлам или реестру и протоколируют те из них, которые попросил запомнить пользователь. Мониторы

обычно состоят из двух частей: драйвера, устанавливаемого в ядро операционной системы, и интерфейсной части, посредством которой пользователь имеет возможность управлять работой монитора и получать результаты протоколирования. Наиболее известными являются, наверное, программы File System Monitor и Registry Monitor, разработанные компанией Sysinternals.

Однако программы могут противодействовать мониторам. Очень важен тот факт, что монитор является активным инструментом — для того чтобы монитор выполнял свои функции, он должен находиться в памяти во время работы исследуемой программы. Следовательно, защищенная программа может обнаружить присутствие монитора и скорректировать свое выполнение разными способами. Например, она может просто отказаться работать, если в памяти присутствует монитор. Другой способ заключается в посылке интерфейсной части монитора сообщения о необходимости завершения работы. При этом монитор оказывается выгруженным из памяти, программа продолжает функционирование в чистом окружении. Красиво выглядит и следующий способ: защищенная программа посылает драйверу монитора команду временно приостановить регистрацию событий, выполняет важные обращения, а затем снова разрешает драйверу работать. При этом интерфейсная часть монитора никак не отражает тот факт, что работа драйвера была откорректирована. И пользователь находится в полной уверенности, что программа не производила никаких обращений, в то время как они имели место, но монитор в это время просто был отключен.

Противодействие мониторам работает только в том случае, если программа знает, как определить наличие монитора в памяти и как его обезвредить. Поэтому часто противнику достаточно изменить логическое имя драйвера монитора, чтобы программа оказалась не в состоянии его обнаружить.

Кроме программ-мониторов, принадлежащих к семейству активных инструментов, есть и семейство пассивных инструментов. Это программы, которые позволяют отслеживать произошедшие изменения, не находясь все время в памяти. Просто до первого запуска защищенной программы необходимо сохранить текущее состояние реестра или определенных файлов, а после запуска сравнить новое состояние с предыдущим. Те записи, которые были изменены, сразу окажутся на подозрении у противника. Использование пассивных инструментов не может быть обнаружено защитой, и единственный способ противостоять им — внести очень большое количество фиктивных изменений, чтобы затруднить противнику определение действительно важных записей. Но такой подход неминуемо приведет к снижению производительности.

14.3.3. Программы с возможностью регистрации

Программы, в которых предусмотрена возможность регистрации, выглядят привлекательно с точки зрения маркетинга. Действительно, пользователей должно радовать, что сразу после оплаты стоимости лицензии они получат регистрационный код и могут моментально превратить ограниченную версию в полноценную. Не потребуется ни выкачивание другого инсталляционного пакета, ни повторная инсталляция.

Однако интуитивно понятно, что подобная схема может быть уязвлена множеством способов.

Очень часто регистрация подразумевает ввод имени пользователя и одного или нескольких регистрационных кодов. Если регистрация не имеет машинозависимой составляющей, т. е. на любом компьютере определенному имени пользователя соответствуют всегда одни и те же регистрационные коды, то, зарегистрировав программу один раз, ее можно будет использовать на любом количестве компьютеров. Неудивительно, что для очень многих программ в Интернете можно без труда найти регистрационные коды.

Проверка правильности введенного регистрационного кода тоже может выполняться по-разному. Нередко встречаются реализации, когда программа вычисляет для введенного имени пользователя правильный регистрационный код, который просто сравнивается с тем кодом, который ввел пользователь. В подобной ситуации для нелегальной регистрации программы на любое имя достаточно ввести это имя и произвольный код, найти точку, где правильный регистрационный код уже вычислен, и прочитать его из памяти. При следующей попытке регистрации достаточно ввести то же самое имя и прочитанный из памяти код, чтобы программа решила, что она корректно зарегистрирована.

И, конечно же, если регистрационные данные просто проверяются на корректность, ничто не мешает противнику исправить тело процедуры проверки таким образом, что любые введенные регистрационные коды будут считаться правильными. Лучше сделать так, чтобы регистрационная информация использовалась в жизненно важных функциях программы.

Это особенно полезно в свете того, что программа после регистрации должна превратиться в полноценную версию без использования каких-либо дополнительных модулей. Следовательно, ограниченная версия должна в какой-либо форме содержать весь код, доступный в полной версии. Одно из возможных решений — зашифровать фрагменты программы, относящиеся к полной версии, а ключ шифрования каким-то образом связать с регистрационным кодом. При этом без знания правильной регистрационной ин-

формации не удастся расшифровать защищенные фрагменты, даже если отключить все проверки правильности регистрационного кода.

14.4. Распределенные проверки

Когда программа получила от пользователя регистрационную информацию, не обязательно сразу же определять ее правильность и информировать об этом пользователя. Если все проверки сосредоточены в одном месте, противнику гораздо легче определить, откуда начинать анализ.

Предпочтительнее выглядит следующий подход. При вводе регистрационной информации выполняется первичная проверка, предназначенная скорее для исключения ошибок набора с клавиатуры, чем для защиты. Если введенная информация кажется правильной, она сохраняется на диске (в файле или реестре) и пользователю выдается сообщение с благодарностью за регистрацию и предложением перезапустить программу, т. к. только после перезапуска будут активированы все функции, недоступные в бесплатной оценочной версии.

После перезапуска программа читает с диска регистрационную информацию и тиражирует ее в памяти в нескольких экземплярах. Это делается для того, чтобы противнику сложнее было найти место, где программа интерпретирует регистрационные данные.

В разных местах программы необходимо вставить разные функции, которые будут проверять разные кусочки регистрационной информации или целостность программы. Лучше иметь несколько функций, проверяющих одно и то же, чем одну функцию, проверяющую сразу все. Противнику придется разбираться со всеми функциями проверки, и чем их больше, тем ему будет сложнее.

Если в результате одной из проверок выяснилось, что регистрационные данные неверны, не стоит сразу сообщать об этом пользователю. Лучше установить специальный флаг, указывающий на то, что программа не была корректно зарегистрирована. А в другой функции, относящейся к совершенно иному фрагменту программы, анализировать один или несколько флагов и предпринимать соответствующие действия.

Действия могут быть самые разнообразные. Так, например, один из DOS-овских симуляторов Formula-1 при запуске проверял наличие документации: у пользователя просили ввести слово, написанное на определенной странице. Исправлением одного байта можно было заставить программу запускаться при вводе любого слова, но тогда автомобиль становился неуправляемым через несколько минут после начала гонки.

Другой пример нестандартной реакции. Программа ReGet Deluxe, предназначенная для управления выкачиванием файлов по протоколам HTTP и FTP, иногда подменяла загруженные файлы, если обнаруживала, что код программы был модифицирован. Так у пользователя взломанной версии ReGet Deluxe были шансы в архиве обнаружить файл readme.txt, содержащий текстовую строку "This file downloaded with cracked version of ReGet Deluxe", или получить сообщение с тем же текстом при запуске закачанного исполняемого файла.

Также существует красивая легенда, касающаяся программы 1С:Бухгалтерия.

1 С:Бухгалтерия и ключи HASP

Бухгалтерская программа от компании 1С исторически была защищена от копирования при помощи аппаратных ключей HASP. Для всех ключей HASP того времени существовали эмуляторы, и 1С нормально работала при отсутствии ключа, если на машине был установлен эмулятор, и очень многие это знали. Но, видимо, о существовании эмуляторов были неплохо осведомлены и программисты 1С. Во всяком случае, однажды программа перестала работать под эмулятором, в то время как при наличии настоящего ключа никаких проблем не возникало.

В обычное время нелегальные пользователи постарались бы найти другой эмулятор или взломанную версию программы, но так случилось, что программа перестала работать за несколько дней до срока сдачи очередного баланса. В результате, очень многие не колебались и предпочли приобрести легальную версию программы вместо того, чтобы платить штраф за просроченный баланс.

Расчет был сделан очень точно, и, по слухам, компания 1С за 2 недели продала столько копий своей Бухгалтерии, сколько обычно продавалось за полгода.

Вот так небольшая задержка в реакции на нарушение защиты, совмещенная со знанием особенностей рынка, помогла быстро легализовать большое количество пользователей.

14.5. Инсталляторы с защитой

Некоторые производители программного обеспечения позволяют всем желающем скачать инсталлятор программы с официального интернет-сайта. При этом инсталлятор может быть защищен таким образом, что для установки программы необходимо знать некоторую информацию, которую производитель сообщит только после получения оплаты.

Однако далеко не все разработчики знают, что некоторые способы защиты инсталляторов совсем не так безопасны, как хотелось бы.

14.5.1. ZIP-архивы с паролем

Очень многие программы распространяются в ZIP-архивах. И очевидная идея защиты дистрибутива — установка пароля на архив. Если выбранный пароль содержит буквы, цифры и знаки препинания и имеет достаточно большую длину, то для отыскания такого пароля перебором потребуется очень много времени. Однако кроме подбора пароля методом грубой силы (brute force) существуют и более эффективные варианты атаки.

Если архив был создан с помощью программы, основанной на исходных текстах от InfoZIP Group (например WinZip), и содержит пять или более файлов, существует алгоритм, отыскивающий ключ и расшифровывающий архив примерно за час.

Если применялся архиватор, не основанный на InfoZIP, то противник может попытаться воспользоваться атакой на основе открытого текста. Для этого ему понадобится иметь в открытом виде один из файлов, зашифрованных в архиве.

Но как можно получить незашифрованный файл? Оказывается, создатели архива часто сами помещают в архив с дистрибутивом файл, который можно найти в открытом виде. Например, файл README.TXT, описывающий продукт, обычно доступен по ссылке с web-страницы, а также находится в дистрибутиве.

Если инсталлятор программы, зашифрованный в архиве, создан при помощи пакета InstallShield, то в архиве окажется с десяток файлов, некоторые из которых идентичны для всех инсталляторов, созданных с помощью InstallShield той же версии. Следовательно, незашифрованный файл может быть взят от другой программы или найден в Интернете с помощью службы FTPsearch.

14.5.2. Norton Secret Stuff

Некоторые производители для распространения своих продуктов использовали разработанную компанией Symantec бесплатную программу Norton Secret Stuff (NSS), создающую из указанного набора файлов самораспаковывающийся архив с паролем. Все данные шифруются с помощью криптографически стойкого алгоритма BlowFish, но в силу действовавших на момент создания NSS ограничений на экспорт программного обеспечения, использующего стойкую криптографию, применяется ключ шифрования, в котором неизвестными являются только 32 бита.

В 1998 году Павел Семьянов разработал программу No More Secret Stuff (NMSS), позволяющую подобрать 32-битовый ключ и расшифровать архив не более чем за 4 недели на компьютере с процессором Intel Pentium 166 МГц. С тех пор производительность процессоров значительно выросла, и подбор ключа может быть осуществлен на одном компьютере за считанные дни. А если выполнять вычисления одновременно на нескольких машинах, то результат может быть достигнут буквально за насколько часов.

Ключ шифрования вычисляется из пароля при помощи *хэш-функции* MD5. На настоящий момент не существует эффективного способа обращения этой функции, но на подбор пароля, порождающего заданный 32-битовый ключ, уходит примерно в 64 раза меньше времени, чем на поиск самого ключа. Таким образом, несмотря на то, что поиск пароля не является необходимым для расшифрования архива NSS, если известен ключ шифрования, при желании можно найти и пароль, затратив при этом на 2 % больше ресурсов.

14.5.3. Package For The Web

Если инсталлятор состоит не из одного файла, часто применяется дополнительная программа, позволяющая все файлы, составляющие инсталлятор, упаковать в один исполняемый файл. Пользователь скачивает этот файл и запускает его, что приводит к распаковке инсталлятора во временную директорию и запуску процесса инсталляции. После того как инсталляция завершится, автоматически будут удалены все временные файлы.

Популярной программой для упаковки инсталляционного пакета в один исполняемый файл является Package For The Web (PFTW), бесплатно распространяемая InstallShield Software Corporation и, похоже, входящая в состав коммерческой программы InstallShield. PFTW позволяет установить пароль на исполняемый файл, что не позволит распаковать, а значит, и запустить инсталлятор, пока не будет введен правильный пароль.

Однако то, как проверяется пароль в PFTW, заслуживает серьезной критики. В ранних версиях пароль просто сравнивался со строкой, хранившейся в зашифрованном виде. Следовательно, можно было или откорректировать условие, в результате проверки которого принималось решение о правильности ввода пароля, или "подсмотреть" пароль в момент проверки.

Более новые версии PFTW поступают гораздо умнее. Пароль нигде не хранится даже в зашифрованном виде, но запоминается его 14-битовый хэш, который сравнивается со значением хэша от введенного пользователем пароля. В случае несовпадения пользователя сразу информируют о том, что пароль неверен. Но так как длина хэша составляет всего 14 бит, примерно один из 16 тысяч паролей будет проходить эту проверку. Это широко рас-

пространенный подход, который позволяет защититься от случайных опечаток, но не дает возможности правильно определить пароль, даже если удастся обратить хэш — каждому значению хэша будет соответствовать, например, почти полмиллиона семисимвольных паролей, состоящих только из маленьких латинских букв, но только один из этих паролей будет правильным.

Если проверка хэша прошла успешно, из пароля вычисляется ключ, который используется для расшифрования инсталлятора. Но именно тут разработчики PFTW допустили оплошность.

Дело в том, что длина ключа шифрования соответствует длине пароля, и преобразование пароля в ключ является обратимым, т. е., зная ключ шифрования, легко вычислить пароль. А алгоритм шифрования не устойчив к атаке на основе открытого текста, т. е., зная несколько байт открытого текста и соответствующего ему шифртекста, очень просто вычислить фрагмент ключа шифрования той же длины, что и известный открытый текст. Таким образом, для определения пароля достаточно найти фрагмент открытого текста, который по длине не короче пароля.

Но, как оказалось, шифруемые данные представляют собой так называемый CAB-файл, формат которого был разработан корпорацией Microsoft. CAB-файл является разновидностью архивного файла, хранящего один или несколько других файлов в упакованном виде, и всегда начинается со строковой сигнатуры "MSCF" (Microsoft CAB File), за которой следуют 4 нулевых байта и 64-битовое число, соответствующее размеру CAB-файла. Таким образом, определить первые 16 байт открытого текста, а значит и пароля, длина которого не превышает 16 байт, не составит труда. В CAB-файле присутствуют и другие области, значение которых совсем не трудно предугадать. Их можно использовать для вычисления более длинных паролей.



Часть IV

ОСНОВНЫЕ АСПЕКТЫ ЗАЩИТЫ ДАННЫХ

В этой части книги собран материал, относящийся к обеспечению безопасности данных. Следующие четыре главы рассказывают о приемах и ошибках в обеспечении секретности, о возможных подходах к реализации систем управления цифровыми правами и применении стеганографии для защиты данных. Также приводится анализ причин появления ненадежных средств защиты.



Глава 15

Обеспечение секретности

Практически всегда под защитой данных понимается наложение ограничений на доступ к информации. Эти ограничения, как правило, бывают трех типов:

- кто имеет право доступа;
- в течение какого периода разрешен доступ;
- какие виды доступа разрешены.

Пользователь может взаимодействовать с данными двумя основными способами: локально и удаленно. При удаленном (сетевом) доступе пользователь сначала проходит процедуры идентификации и аутентификации, а потом запрашивает у сервера необходимую ему информацию. Сервер выполняет все необходимые проверки и принимает решение либо о предоставлении доступа, либо об отказе. Для защиты удаленных данных разработаны формальные модели разграничения доступа, описание которых можно найти в любом хорошем учебнике по сетевой безопасности.

Мы же будем рассматривать ситуации, когда все проверки реализуются локально. То есть пользователь (или противник) теоретически может полностью контролировать все шаги, которые та или иная программа выполняет для получения доступа к данным.

В условиях локальности проверок ограничивать, например, период доступа гораздо сложнее, чем при выполнении контроля на сервере. Но для простого обеспечения секретности, по идее, нет никаких препятствий — достаточно спросить у пользователя пароль или получить от него какую-то другую аутентифицирующую информацию, при помощи хорошей хэш-функции вычислить ключ шифрования и зашифровать на нем защищаемые данные при помощи стойкого криптографического алгоритма. Однако в практических реализациях этой простейшей последовательности действий разработчики ухитряются допускать самые разнообразные ошибки, значительно снижающие уровень защищенности информации, а иногда и вовсе сводят его к нулю.

Защита данных может осуществляться в совершенно разных условиях, и, соответственно, в каждом случае необходимо использовать свои приемы защиты.

15.1. Архивация с шифрованием

Многие современные программы упаковки данных (архиваторы) имеют встроенную поддержку шифрования. Если пользователь пожелает защитить от чужих глаз информацию, находящуюся в архиве, ему надо при упаковке ввести пароль, а архиватор сам выполнит все остальные действия. При попытке извлечения зашифрованного файла архиватор запросит у пользователя пароль и распакует файл только в том случае, если пароль верен.

Стоит отметить, что зашифрование всегда выполняется после компрессии, т. к. зашифрованные данные должны быть неотличимы от случайной последовательности, и, следовательно, архиватор не сможет найти в них избыточность, за счет удаления которой и происходит упаковка.

15.1.1. ZIP

Свойства алгоритма шифрования, использованного в архивном формате ZIP, уже многократно обсуждались в этой книге.

Не повторяя детали, вспомним, что алгоритм шифрования уязвим к атаке на основе открытого текста и достаточно знать 12 незашифрованных байт (после компрессии), чтобы расшифровать весь файл за приемлемое время.

Для архивов, содержащих 5 и более файлов и созданных архиваторами, основанными на библиотеке InfoZIP, возможна атака, использующая в качестве открытого текста данные из заголовков зашифрованных файлов. Этой атаке были подвержены файлы, созданные при помощи WinZIP, но в последних версиях этого архиватора проблема была исправлена.

Еще в июле 2002 года компания PKWARE, Inc. выпустила версию архиватора PKZIP, поддерживающего более стойкие алгоритмы шифрования. Но, видимо, по той причине, что PKZIP позиционируется как продукт для корпоративных пользователей, новое шифрование не получило широкого распространения.

15.1.2. ARJ

Популярный во времена DOS, но довольно редко используемый в настоящее время архиватор, разработанный Робертом Янгом (Robert Jung), использовал следующий алгоритм шифрования.

Из пароля по очень простому обратимому алгоритму получалась гамма, равная по длине паролю. Эта гамма накладывалась на шифруемые данные путем сложения по модулю 2 (операция XOR). Таким образом, наличие открытого текста, равного по длине паролю, позволяло моментально определить гамму и использованный пароль.

Более того, в самом начале упакованных данных содержалась информация компрессора, такая как таблицы Хаффмана (Huffman tables), и часть этой информации могла быть предсказана, что позволяло значительно повысить скорость поиска пароля перебором.

Начиная с версии 2.60 (ноябрь 1997 года) ARJ поддерживает шифрование по алгоритму ГОСТ 28147-89 *(см. разд. 6.1)*.

15.1.3. RAR

Архиватор, разработанный Евгением Рошалем (Eugene Roshal), является неплохим примером того, как можно подходить к шифрованию данных.

В алгоритме шифрования, используемом в RAR версии 1.5, есть некоторые недочеты. Так эффективная длина ключа шифрования составляет всего 64 бита, т. е. перебором 2^{64} вариантов ключа можно гарантированно расшифровать пароль. Более того, наличие открытого текста позволяет уменьшить множество перебираемых вариантов до 2^{40}. Следовательно, атака может быть успешно выполнена даже на одном компьютере. Скорость перебора на компьютере с процессором Intel Pentium III 333 МГц составляет примерно 600 000 паролей в секунду.

При переходе к версии 2.0, видимо, была проведена серьезная работа над ошибками. Во всяком случае, взлом нового алгоритма шифрования перебором требует примерно 2^{1023} операций, что намного больше, чем может быть выполнено на современной технике. Об эффективных атаках, использующих открытый текст, ничего не известно. Скорость перебора паролей снизилась примерно до 2000 штук в секунду (в 300 раз).

Но разработчики RAR решили не останавливаться на достигнутом. В версии 3.0, появившейся в мае 2002 года, для шифрования стал использоваться алгоритм AES (Rijndael) с ключом длиной 128 бит. Такое решение выглядит вполне разумным как минимум по двум причинам. Во-первых, безопаснее использовать проверенный и хорошо зарекомендовавший себя алгоритм, чем нечто самодельное, и у AES здесь нет конкурентов. А во-вторых, у AES скорость шифрования выше, чем у алгоритма, использованного в RAR 2.0.

Кроме замены алгоритма шифрования, в RAR 3.0 используется и другая процедура получения ключа шифрования из пароля. Эта процедура требует вычисления хэш-функции SHA1 262 144 раза, что позволяет перебирать около 3-х паролей в секунду, т. е. в 600 раз меньше, чем для RAR 2.0.

15.2. Секретность в реализации Microsoft

Уже на протяжении многих лет программы, входящие в Microsoft Office, позволяют зашифровывать документы по паролю, вводимому пользователем. Однако далеко не всегда данные оказываются защищены должным образом.

15.2.1. Microsoft Word и Excel

Шифрование файлов было реализовано уже в Microsoft Word 2.0. Из пароля с помощью легко обратимого алгоритма получалась 16-байтовая гамма, которая накладывалась на содержимое документа. Но вычисление гаммы без пароля не составляло никакого труда, т. к. гамма накладывалась и на служебные области, которые имели фиксированное значение во всех документах.

В Word 6.0/95 и Excel 5.0/95 алгоритм шифрования не претерпел значительных изменений, но изменился формат файлов — он стал основываться на хранилище OLE Structured Storage. Для восстановления пароля документа все также требовалось найти 16-байтовую гамму, использованную для шифрования данных.

Один из методов, позволяющих отыскать гамму, например, для документов Word, основывается на простейшем статистическом анализе. Самый часто встречающийся символ в тексте на любом языке — пробел. Таким образом, достаточно определить код наиболее используемого символа в каждой из 16 позиций, соответствующих разным байтам гаммы. Выполнив операцию XOR каждого найденного значения с кодом пробела (0x20), мы получим 16 байт гаммы.

В программах Word 97/2000 и Excel 97/2000 данные шифруются при помощи алгоритма RC4 с ключом длиной 40 бит. Такое шифрование уже не позволяет моментально определить пароль. Но производительность вычислительной техники за последние годы выросла настолько сильно, что единственно правильный ключ шифрования документа Word (из 2^{40} возможных) может быть найден максимум за четверо суток на компьютере с двумя процессорами AMD Athlon 2600+.

Начиная с Office XP, наконец появилась поддержка шифрования документов ключами длиной более 40 бит. Но, похоже, большинство пользователей до сих пор продолжает использовать 40-битовое шифрование, т. к. оно позволяет открывать защищенные документы в предыдущих версиях офисных программ. Да и изменение настроек шифрования требует дополнительных действий со стороны пользователя (открытия диалога настроек и выбора нужного криптопровайдера), а по умолчанию применяются 40-битовые ключи.

15.2.2. Microsoft Access

Базы данных Microsoft Access могут иметь два типа паролей: пароли на открытие и пароли для разграничения доступа на уровне пользователя.

Пароль на открытие, похоже, никогда не представлял серьезной защиты, т. к., начиная с Access версии 2.0, он хранился в заголовке базы данных. Правда, сам заголовок был зашифрован алгоритмом RC4, но это не сильно увеличивало стойкость, т. к. в рамках одной версии формата всегда использовался один и тот же 32-битовый ключ шифрования, жестко прошитый в динамически загружаемую библиотеку, отвечающую за работу с файлом базы данных.

А учитывая то, что RC4 — синхронный потоковый шифр, достаточно было один раз найти гамму, порождаемую RC4 с известным ключом. После этого пароль можно было определить, выполнив сложение по модулю 2 гаммы и нужных байт заголовка.

Так для Access 97 необходимо было выполнить XOR 13 байт, расположенных по смещению 0x42 от начала файла базы данных, со следующей последовательностью:

`0x86, 0xFB, 0xEC, 0x37, 0x5D, 0x44, 0x9C, 0xFA, 0xC6, 0x5E, 0x28, 0xE6, 0x13.`

Альтернативный способ снятия пароля заключался в исправлении заголовка и установке значения байта, соответствующего первому символу, в такое состояние, чтобы после расшифровки заголовка там оказывался нулевой байт. Тогда Access, интерпретирующий пароль как строку, заканчивающуюся нулевым символом, увидев ноль в первом байте, решит, что пароль вообще не установлен. Для снятия пароля в Access 97 необходимо установить байт по смещению 0x42 от начала файла в значение 0x86.

Кстати, разработчики одной коммерческой программы, позволяющей восстановить забытые пароли к базам Microsoft Access, в описании новшеств очередной версии указали, что время, затрачиваемое на отыскание пароля, уменьшилось в 1,5 раза. Вероятно, для этого им пришлось уменьшить задержку перед выводом найденного пароля на экран, т. к. значительно сложнее придумать очень медленный способ выполнения XOR, а потом ускорить его в 1,5 раза.

Начиная с Access 2000, простое наложение гаммы уже не позволяет сразу же определить пароль, т. к. необходимо выполнить еще несколько дополнительных несложных действий. Но пароль все равно хранится в заголовке, а значит, может быть оттуда прочитан.

Что самое занимательное, установка пароля на открытие базы данных не приводит к шифрованию ее содержимого. Однако Access поддерживает такую операцию, как шифрование базы данных, но пароль в этом шифро-

вании никак не участвует, а ключ шифрования хранится в заголовке файла . базы.

Другой тип паролей, поддерживаемых Microsoft Access, используется не для обеспечения секретности, а для разграничения доступа. Но при проектировании, похоже, было допущено несколько ошибок, связанных и с этими паролями.

Правильно было бы хранить не пароли, а их хэши. Но, по непонятным причинам, в системной базе данных, содержащей имена, пароли и прочие атрибуты всех пользователей, можно найти сами пароли, зашифрованные трехкратным DES с двумя ключами в режиме EDE (Encrypt-Decrypt-Encrypt), когда первый ключ применяется дважды, на первом и третьем шаге. Ключи, как обычно, являются константами и хранятся в динамически загружаемой библиотеке. Такая защита позволяет быстро определить пароль любого пользователя, хотя Microsoft утверждает, что утерянные пароли пользователей Access не могут быть восстановлены.

В системной базе данных для каждого пользователя хранится и уникальный идентификатор, являющийся функцией от имени пользователя и некоторой произвольной строки, вводимой при создании учетной записи. И именно этот идентификатор является ключом, по которому идентифицируются пользователи в основной базе данных.

Так, например, у каждой таблицы в основной базе данных есть владелец, который имеет максимальные права. Но в основной базе данных хранится только идентификатор пользователя, являющегося владельцем, а имя и вся вспомогательная информация для аутентификации пользователя хранится в системной базе данных. И создается впечатление, что если системная база данных будет утеряна, то доступ к содержимому основной базы данных получить не удастся.

Но функция вычисления идентификатора пользователя сравнительно легко обращаема, что позволяет определить имя владельца идентификатора и строку, введенную при создании его учетной записи. После этого остается только создать новую системную базу данных и добавить в нее пользователя с известными атрибутами, но вообще без пароля.

15.2.3. Microsoft Money

Программа учета личной финансовой истории Microsoft Money основывае ся на том же ядре базы данных, что и Microsoft Access. Поэтому на протя жении многих версий файлы Money поддерживали точно такой же парол на открытие, как и базы Access.

Однако в последних версиях, начиная с Money 2002, также называемой Money 10, пароль используется для вычисления ключа и последующего шифрования данных алгоритмом RC4, а не просто проверяется путем сравнения с сохраненным значением. То есть пароль может быть найден только подбором.

Но и тут не обошлось без "оригинальных" технических решений. Дело в том, что зашифровывается не весь файл, а только 15 первых блоков (в новых версиях каждый блок занимает 4 Кбайта). Такой подход, скорее всего, выбран с целью обеспечения возможности создания компактных архивных копий баз Money.

Действительно, файл может иметь размер в десятки мегабайт, но если он весь будет зашифрован, ни один архиватор не будет в состоянии уменьшить объем занимаемого файлом дискового пространства. Если же зашифрован только заголовок, то файл очень хорошо упаковывается и при этом не может быть открыт в Money, т. к. важная информация, касающаяся структуры таблиц и расположения их в файле, оказывается недоступной.

Однако подобное решение нельзя назвать правильным. Основная часть данных оказывается в базе в незашифрованном виде и, при желании, может быть легко извлечена противником.

15.2.4. Encrypted File System

Начиная с Windows 2000 операционные системы, основанные на ядре NT, поддерживают Encrypted File System (EFS) — расширение файловой системы NTFS (New Technology File System), позволяющее хранить файлы пользователей в зашифрованном виде. При этом шифрование выполняется совершенно прозрачно и не требует от пользователя никаких дополнительных усилий, кроме однократного указания, что файл должен быть зашифрован.

Даже если противник сможет получить физический доступ к файловой системе и прочитать защищенный файл, ему потребуется получить ключ шифрования для извлечения содержимого.

Симметричный ключ, которым зашифровывается файл (File Encryption Key, FEK), сам зашифрован на открытом ключе, принадлежащем пользователю, имеющему права на доступ к файлу. FEK хранится вместе с зашифрованным файлом, и для его расшифровки используется секретный ключ пользователя.

С каждым файлом может быть ассоциировано несколько копий FEK, зашифрованных на открытых ключах так называемых агентов восстановления данных (Recovery Agents).

Процедура получения всей необходимой для расшифровки информации включает в себя много этапов. Однако в Windows 2000 реализация EFS такова, что в большинстве случаев все зашифрованные файлы могут быть извлечены без знания пароля владельца или агента восстановления.

15.3. Шифрование дисков

Еще во времена DOS появились программы, позволяющие создавать защищенные диски, содержимое которых становилось доступным только после того, как пользователь вводил правильный пароль.

На первый взгляд, защитить информацию в подобных условиях не очень сложно, но разработчики часто не справлялись с поставленной задачей.

15.3.1. Stacker

Одной из первых программ, позволявших защищать диски, была программа Stacker, разработанная компанией Stac Electronics, Inc. Точнее, Stacker предназначался для создания сжатых дисков, поддерживающих упаковку и распаковку информации "на лету". Но одна из опций позволяла защитить хранимую на диске информацию паролем.

Однако в программе использовалась плохая *хэш-функция*, вследствие чего сложность подбора пароля, пригодного для расшифровки диска, была всего 2^s, т. е. пароль подбирался моментально.

15.3.2. Diskreet

Одна из программ, входивших в состав пакета Norton Utilities, называлась Diskreet и была предназначена для создания зашифрованных файлов и дисков.

Diskreet поддерживал два метода шифрования, один из который основывался на DES и работал очень медленно, а другой, самодельный и более быстрый, так и описывался в документации: "fast, proprietary method" (быстрый, патентованный метод).

Этот "патентованный" метод оказался очень примитивным и неустойчивым к атаке на основе открытого текста. А разработчики, похоже, приложили максимум усилий для того, чтобы открытый текст было легко найти — зашифрованные файлы изобиловали фрагментами с предопределенными значениями.

А согласно информации, приведенной на страничке "Russian Password Crackers" (**www.password-crackers.com/crack.html**), созданной Павлом Семьяновым, пароль от диска, созданного при помощи Diskreet из Norton

Utilities 8.0, просто хранится в файле DISKREET.INI в слегка модифицированном виде и может быть извлечен при помощи программы, приведенной в следующем алгоритме (листинг 15.1).

Листинг 15.1. reetpsw.c • • вычисление пароля Diskreet

```
#include <stdio.h>
void main (void) {
  unsigned char b, bXor;
  FILE *t = fopen ("c:\\nu\\DISKREET.INI", "rb");
  fseek (f, 0x64L, SEEK_SET);
  for (bXor = 0x35; b = fgetc (f); bXor += 0x36) {
    if ((b ^=bXor) == 0) b = 0x33;
    putchar (b);    // здесь выводятся символы пароля ;-)))
  }
  fclose (f);
```

15.3.3. BootLock

Еще один программный продукт, предназначенный для защиты данных и выпускавшийся, как и Norton Utilities, компанией Symantec, назывался Norton You Eyes Only (NYEO). Одной из составляющих NYEO являлась программа BootLock, позволявшая зашифровать даже загрузочный диск, сделав, таким образом, всю вычислительную систему недоступной для противника.

Однако разработчики BootLock по неизвестным причинам приняли решение шифровать не все данные диска, а только системные области: загрузочную запись (Boot) и корневую директорию (Root). А таблица размещения файлов (File Allocation Tables, FAT) и область данных, в которой хранится содержимое файлов, оказывались незашифрованными.

Шифрование выполнялось путем наложения 512-байтовой гаммы на каждый шифруемый сектор. Причем для всех секторов одного диска использовалась одна и та же гамма. Следовательно, наличие одного открытого сектора давало возможность расшифровать все остальные сектора.

Одной из особенностей файловой системы FAT является то, что количество записей в корневой директории определяется при форматировании диска и не может быть ни увеличено, ни уменьшено. Пока на диске не создано ни

одного файла, все сектора, относящиеся к корневой директории, заполнены нулями. Информация, описывающая каждый добавленный файл или директорию, занимает 16 байт.

Обычно в корневой директории резервируется место под несколько сотен файлов, но редко кто хранит много файлов в корне диска. Следовательно, с очень большой вероятностью в последнем секторе корневой директории за все время не будет ни одной записи, т. е. сектор будет содержать они нули. А сектор, зашифрованный при помощи BootLock, — гамму, использованную для шифрования.

Причем складывается ощущение, что компания Symantec знала о нестойкости защиты, обеспечиваемой ее программным продуктом. Во всяком случае, на сайте компании можно было найти информацию о платной услуге по восстановлению данных диска, защищенного BootLock, в случае утери пароля. Стоимость этой услуги составляла всего $ 300 при восстановлении в недельный срок или $ 600 при восстановлении за 24 часа.

15.4. Документы PDF

Формат переносимых файлов PDF (Portable Documents Format), разработанный компанией Adobe Systems, Inc., предназначен для хранения документов, представленных в электронной форме. PDF является основным форматом продуктов семейства Acrobat.

Во второй версии спецификации (PDF 1.1) была добавлена поддержка шифрования, но начальная редакция алгоритма защиты оказалась не очень удачной — похоже, один и тот же ключ использовался многократно при шифровании потоковым алгоритмом RC4. Точного описания особенностей реализации этого алгоритма найти не удалось, но официальная документация от Adobe гласит, что этот алгоритм не поддерживается и не рекомендуется к использованию.

Для более гибкого управления процедурой вычисления ключа шифрования продукты семейства Acrobat поддерживают так называемые модули защиты (Security Handlers), которые могут быть подключены через средства расширения (plug-ins). Изначально Security Handler отвечал только за вычисление ключа шифрования документа, а все операции по шифрованию выполнял сам Acrobat.

Вплоть до версии Acrobat 5.0 ключ шифрования документов по официальной информации всегда имел длину в 40 бит из-за экспортных ограничений. Однако уже в Acrobat Reader 4.0.5 появилась поддержка ключей большей длины, использовавшихся для защиты электронных книг.

Начиная с Acrobat 5.0 (и соответствующей ему спецификации PDF 1.4) были добавлены сразу две новых версии алгоритма защиты, позволяющие работать с ключами длиной до 128 бит. Одна из новых версий была документирована, а вторая формально держится в секрете по требованию департамента коммерции США (хотя ее поддержка давно реализована в продуктах сторонних компаний).

Наконец, в Acrobat 6.0 (спецификация PDF 1.5) модуль защиты получил возможность не только вычислять ключ шифрования, но и выполнять само шифрование.

Возможность создания собственных модулей защиты с нужной функциональностью привела к тому, что на рынке появилось несколько таких модулей, разработанных разными компаниями. Рассмотрим основные характеристики некоторых из них.

15.4.1. Password Security (Standard Security Handler)

Этот модуль защиты был разработан компанией Adobe и является основным средством защиты, встроенным в Acrobat и бесплатный Acrobat Reader.

При защите документов стандартным методом автор имеет возможность установить два пароля: для владельца и пользователя соответственно. Для того чтобы открыть документ и отобразить его на экране, достаточно знать любой из паролей, но некоторые операции, такие как редактирование документа, его печать или копирование фрагментов в буфер обмена, могут быть недоступны, если введен пароль пользователя. И, разумеется, настройки защиты могут быть изменены только в том случае, если документ открыт по паролю владельца.

Часто на документ устанавливается только пароль владельца, а пароль пользователя остается пустым. При этом документ открывается без запроса пароля, но операции, запрещенные автором, оказываются недоступными.

Очевидно, что если известен хотя бы один пароль или пароль пользователя просто отсутствует, то документ может быть расшифрован и сохранен со снятыми ограничениями. Это нельзя считать ошибкой реализации — это просто свойство, о котором стоит помнить.

Грубых ошибок в реализации стандартной защиты допущено не было, но несколько недочетов, относящихся к самой популярной версии алгоритма, использовавшей только 40-битовый ключ, найти можно.

Так процедура проверки пароля выполняется очень быстро. Для проверки правильности пароля пользователя необходимо вычислить один раз значе-

ние хэш-функции MD5 и расшифровать 32 байта алгоритмом RC4. Проверка пароля владельца требует повторить те же действия дважды. Все это позволяет подбирать пароли с очень высокой скоростью.

Если же проводится поиск 40-битового ключа шифрования, то для проверки правильности ключа требуется лишь одно шифрование алгоритмом RC4 Это позволяет на современном компьютере гарантированно найти клю| примерно за одну неделю.

То, как проверяется правильность ключа шифрования, позволяет значительно сократить среднее время, затрачиваемое на расшифрование одного документа за счет предварительных вычислений и сохранения вспомогательных данных на диске.

Также один раз перебрав 2^{40} ключей, можно расшифровать любое количество документов, защищенных стандартным методом.

Все эти недочеты были исправлены с появлением новой версии алгоритма в Acrobat 5. Так для проверки каждого пользовательского пароля теперь требуется выполнить 51 вычисление MD5 и 20 шифрований RC4. Это снижает скорость проверки паролей в несколько десятков раз. А перебрать 2^{128} ключей шифрования современный уровень развития технологий не позволяет.

15.4.2. Другие модули защиты от Adobe

В Acrobat Reader 4.0.5 появилась поддержка модуля Adobe PDF Merchan (Adobe.WebBuy), предназначенного для защиты электронных книг. Документы защищались ключами длиной от 64 до 128 бит.

Примерно в то же время получил распространение модуль защить EBX_HANDLER, изначально реализованный в программе GlassBook Reader, разработанной компанией GlassBook, Inc. и предназначенной для покупки и просмотра защищенных электронных книг в формате PDF. Позже Adobe приобрела компанию GlassBook, их Reader оказался переименован в Adobe eBook Reader, а EBX HANDLER стал основным модулем защиты для технологии распространения электронных книг, продвигаемой Adobe *(более подробная информация о защите электронных книг в формате PDF приведена в гл. 16)*.

Еще один модуль защиты, разработанный Adobe, назывался Acrobat Self-Sign Security (Adobe.PPKLite) и представлял собой первую попытку использования схем с открытым ключом в защите PDF-документов, предпринятую в Acrobat 5. Однако эта попытка не имела большого успеха, т. к. не опиралась на существующую инфраструктуру открытых ключей.

На смену Self-Sign Security пришел модуль Certificate Security (Adobe.PubSec), представленный в Acrobat 6, который уже умел работать

с сертификатами, выдаваемыми основными мировыми центрами сертификации.

Но оба эти решения относятся скорее к организации защищенного корпоративного документооборота, чем к обеспечению секретности, и подробно рассматриваться не будут. Хотя для обеспечения секретности они вполне пригодны.

15.4.3. SoftLock (SLCK_SoftLock)

Модуль защиты SoftLock, разработанный одноименной компанией, получает 40-битовый ключ шифрования документа следующим образом. При первом открытии документа вычисляется значение, являющееся функцией от идентификатора документа и некоторых параметров конкретного компьютера (например метки тома диска С), и это значение сообщается пользователю. В ответ пользователь должен ввести 8-символьный ключ, полученный от продавца документа. На основании полученного от пользователя ключа и параметров компьютера вычисляется и проверяется ключ шифрования документа. Следовательно, при переносе на другой компьютер документ невозможно будет открыть с тем же 8-символьным ключом, т. к. параметры компьютера, а значит, и вычисленный ключ шифрования документа, будут иными.

Если бы каждый из восьми символов ключа, вводимого пользователем, мог принимать одно из 95 значений ASCII-символов, доступных на стандартной английской клавиатуре, то общее число комбинаций превысило бы 2^{52}. Однако каждый символ преобразуется в одно из 16 возможных значений, а 2 символа из 8 являются контрольными и не участвуют в вычислении ключа шифрования. Таким образом, существует всего 2^{32} различных 8-символьных ключей, и один из них является правильным. Полный перебор 2^{32} ключей, очевидно, не отнимает много времени и позволяет быстро расшифровать любой документ с защитой SoftLock.

На настоящий момент компания SoftLock уже не существует и модуль защиты больше не поддерживается.

15.4.4. Newsstand Crypto (NWST_Crypto)

Модуль NewsStand, разработанный в NewsStand, Inc., используется для защиты периодических изданий, распространяемых через Интернет в электронной форме. Например, всего за $ 0.65 можно приобрести свежий выпуск New York Times, находясь при этом в любой точке Земли, откуда есть доступ к Интернету.

NewsStand использует 40-битовые ключи шифрования, но каждый из пяти байт ключа принимает только 16 возможных состояний — от '0' до '9' и от 'A' до 'F'. Вследствие этого эффективная длина ключа оказывается равна всего 20 битам, а 2^{20} комбинаций перебираются на современном компьютере за считанные секунды.

15.4.5. Panasonic Crypto (PSDS_Crypto)

Этот модуль использовался для защиты сервисной технической документации к аппаратуре, производимой компанией Panasonic. Формально для получения доступа к зашифрованным документам требовался аппаратный ключ защиты, втыкаемый в LPT-порт. Но процедура получения ключа шифрования была реализована не лучшим способом.

Выполнялась только проверка наличия аппаратного ключа без использования хранимой в ключе информации для расшифровки документов. Кроме того, в результате вычисления ключа шифрования в модуле защиты всегд возвращалось одно из двух возможных 40-битовых значений. То есть достаточно было проверить всего 2 ключа для того, чтобы расшифровать любой документ с такой защитой.

15.4.6. KEY-LOK Rot13 (BPTE_rot13)

Этот модуль защиты также требует наличия аппаратного ключа для открытия документов, но, на самом деле, всегда используется один и тот же фиксированный 40-битовый ключ шифрования, который жестко прошит в теле модуля защиты.

Весьма занимательно, что в состав SDK для Adobe Acrobat 4 входит пример модуля защиты с названием Rot13. И в этом примере ключ шифрования также является константой. Видимо, разработчики модуля KEY-LOK Rot13 не осмелились сильно модифицировать готовый пример, а просто вставили проверку наличия аппаратного ключа и изменили константу, используемук в качестве ключа шифрования.

15.4.7. Normex

Существует как минимум четыре модуля защиты, разработанные компаниек Normex, Ltd. Эти модули называются Normex level 1 (NORM_NxSec1), Normex level 2 (NORM_NxSec2), Normex level 3 (NORM_NxSec3) и Internet demo (NORM_NxSecInDemo).

Скорее всего, эти модули защиты чем-то отличаются друг от друга, но их объединяет одно очень важное свойство — для шифрования документа они используют фиксированные 40-битовые ключи.

То есть достаточно один раз определить значения ключей, которые жестко прошиты в файлах, содержащих код модуля зашиты, после чего любой документ можно будет моментально расшифровать.

15.4.8. Liebherr (LEXC_Liebherr_Security)

Модуль защиты, разработанный компанией LexCom Informationssysteme GmbH, также не обеспечивает никакой секретности, т. к. использует фиксированный 40-битовый ключ шифрования для всех документов.

15.4.9. DocuRights

Модуль защиты DocuRights, разработанный компанией Aries Systems Corp., построен несколько нестандартным образом. PDF-документ, с которым имеет дело пользователь, представляет собой контейнер, зашифрованный при помощи стандартного модуля защиты (Standard Security Handler, SSH) с пустым паролем, внутри которого находится другой PDF-документ, но защищенный уже с помощью DocuRights.

Однако такая схема никак не увеличивает защищенность и, скорее всего, используется исключительно для более красивой подачи пользователю информации о том, что у него нет прав доступа к документу. Стойкость защиты все равно определяется тем, как DocuRights вычисляет ключ шифрования. Но здесь разработчики, видимо, решили себя не утруждать, т. к. для шифрования всегда используется постоянный 40-битовый ключ.

15.4.10. FileOpen Publisher (FOPN_fLock)

Этот модуль защиты разработан компанией FileOpen Systems и является составной частью ее продукта FileOpen Publisher, лицензия на который стоит $ 2500.

Согласно рекламе, FileOpen Publisher предназначен для издателей, желающих иметь полный контроль над распространением произведений в электронной форме. Этот продукт включает в себя инструменты для управления электронной подпиской и аутентификацией конкурирующих пользователей, позволяет создавать документы с ограниченным сроком действия и документы, привязанные к конкретным носителям, например CD-ROM. Также издатель имеет возможность ограничивать, когда и в каком объеме можно распечатывать документ.

FileOpen Publisher прошел довольно длинный путь эволюции в области защиты.

Вплоть до версии 2.3 для всех защищаемых документов использовался 40-битовый фиксированный ключ.

В версии 2.4 появилась поддержка переменных ключей, но вся информация, необходимая для вычисления ключа, хранилась в самом документе.

В версии 2.5 (самой новой на настоящий момент) была добавлена возможность ручного ввода ключа шифрования, который будет использоваться для всех документов, защищаемых в рамках текущего проекта. Этот ключ не хранится в документе, но и он не обеспечивает нормальной защиты по нескольким причинам.

Во-первых, у каждого защищенного документа должен быть свой уникальный ключ, и знание ключа к одному документу не должно позволять вычислить ключ к любому другому документу. А это требование не выполняется, т. к. у всех файлов одного проекта будет один и тот же ключ.

Во-вторых, ключ шифрования должен генерироваться случайным образом, т. к. человек обычно вводит предсказуемые данные.

А в-третьих, в диалоговом окне, где издателю предлагают вручную задать ключ, можно вводить только цифры. И первые пять введенных символов будут использованы в качестве ключа шифрования. То есть издатель может ввести всего 10^5 различных ключей, а значит, эффективная длина ключа составит менее 17 бит.

15.4.11. FileOpen WebPublisher (FOPN_foweb)

Этот модуль защиты является составляющей частью еще одного продукта компании FileOpen, носящего имя WebPublisher2 и предназначенного для распространения электронных документов и управления доступом к ним через Интернет.

Первая версия WebPublisher2 в каждом документе сохраняла и ключ шифрования, необходимый для открытия документа. То есть стойкость защиты была нулевой.

После выпуска обновления ключ не сохраняется внутри документа, но его длина составляет всего 40 бит, хотя WebPublisher2 был выпущен в феврале 2002 года, а пятая версия Acrobat SDK, включающая все необходимое для поддержки ключей длиной до 128 бит, появился в апреле 2001 года.

Один и тот же ключ шифрования снова используется для всех документов, защищаемых в рамках одного проекта, т. е. один раз затратив вычислительные ресурсы на поиск ключа, можно будет расшифровать сразу несколько документов.

Ключ снова выбирается не случайно, а задается издателем в файле настроек как ASCII-строка. Это ограничивает множество возможных символов, использующихся в ключе. Да и вряд ли издатель будет вводить истинно случайную комбинацию. Так файлы примеров, выложенные на сайте FileOpen,

были защищены ключами 'abcde', 'bcdef, 'cdefg', 'mnopq' и 'rstuv'. Образец защищенного документа на сайте Briefsmart шифровался по ключу '1lzg1', а компания IPexpert, Inc., распространяющая свои файлы под защитой WebPublisher2, использует в ключе только десятичные цифры.

15.4.12. Другие модули защиты

Кроме уже перечисленных модулей защиты, существуют и другие.

Так, например, модуль Australian Standards Online (SASS INTERNET_STDS) использовался для защиты Австралийских стандартов, распространяемых в электронной форме. Этот модуль не содержал грубых ошибок в реализации защиты. Но в апреле 2001 года в Australian Standards отказались от использования шифрования при распространении документов, введя при этом защиту на основе водяных знаков.

Уровень защиты, эквивалентный 40-битовому ключу, обеспечивали и модули защиты PDFLock (XESC_XELock) и Montrose. Но эти модули тоже давно не поддерживаются (по неизвестным причинам), и найти какую-либо полезную информацию о них уже практически невозможно.

В составе продуктов семейства Acrobat версии 5 поставлялся модуль защиты InterTrust.DocBox, разработанный компанией InterTrust Computing. Но его, вероятно, тоже постигла печальная участь, т. к. в Acrobat 6 он уже не входит. Хотя сама компания InterTrust продолжает существовать и даже обвиняет Microsoft в том, что последняя нарушает 8 патентов, принадлежащих InterTrust.

Еще существует модуль защиты Authentica PageRecall (PageVault), разработанный компанией Authentica, Inc., но он нацелен на корпоративный рынок — стоимость лицензии на PageRecall начинается с S 32 500. И достоверных результатов оценки стойкости защиты, обеспечиваемой PageRecall, пока нет.

Таким образом, из доступных на настоящий момент модулей защиты, разработанных различными компаниями, за исключением Adobe и Authentica, нет ни одного, который бы применял 128-битовое шифрование или хотя бы использовал все ресурсы 40-битового ключа.

И нет никакой надежды, что ситуация в ближайшее время изменится. Если раньше стоимость лицензии на Reader Integration Key, позволяющей использовать модуль расширения в бесплатных версиях Acrobat Reader, составляла $ 100 и позволяла создавать любое количество модулей расширения, работающих в Reader, то с выходом Acrobat 6.0 все изменилось. Теперь за право доступа к полной версии SDK придется заплатить $ 200, a Reader Integration Key на каждый модуль расширения, предназначенный для свободного использования, обойдется в $ 2500.

Но этого мало. Для создания собственного модуля защиты PDF-документов необходима специальная лицензия, которая имеет еще более высокую стоимость.

Разумеется, Adobe имеет право проводить любую маркетинговую политику в отношении своих собственных продуктов, но нынешняя ситуация явно не способствует появлению недорогих и надежных решений для защиты документов в формате PDF.

15.5. Уничтожение информации

Иногда для предотвращения утечки информации часть данных должна быть уничтожена. Чаще всего это требуется в случаях, когда контейнер, хранящий секреты, должен попасть в чужие руки.

Однако людей, которые действительно заботятся о том, чтобы правильно уничтожать информацию, довольно мало.

Название врезки

В январе 2003 года была опубликована статья "Remembrance of Data Passed: A Study of Disk Sanitization Practices" ("Памяти ушедших данных: изучение приемов очистки жестких дисков"), в которой рассматривались вопросы уничтожения данных, хранившихся на жестких дисках.

Авторы статьи, студенты Массачусетского технологического института (Massachusetts Institute of Technology, MIT) Симсон Л. Гарфинкел (Simson L. Garfinkel) и Аби Шелат (Abhi Shelat), приводят весьма печальные результаты проведенных ими исследований.

В период с ноября 2000 года по август 2002 года они приобрели 158 жестких дисков, бывших в употреблении. 129 из них (примерно 82 %) оказались в работоспособном состоянии. И только 12 из них (9 %) были полностью очищены от информации (заполнены нулевыми секторами).

С очень многих дисков удалось восстановить массу интересной информации, такой как медицинские и финансовые данные, личная переписка и т. п. На двух дисках хранились 2868 и 3722 номеров кредитных карт соответственно. Один из этих дисков, предположительно, стоял в банкомате в Иллинойсе.

Файлы с части дисков были удалены средствами операционной системы. Такие файлы, чаще всего, могут быть легко восстановлены специальными программами.

Некоторые диски были отформатированы. Но при обычном форматировании (например с помощью программы FORMAT) затираются только служебные об-

ласти, хранящие информацию о расположении файлов на диске, а также имена и размер файлов в корневом каталоге. Но само содержимое файлов никак не уничтожается. Даже если с диска удаляется логический раздел, это не приводит к физическому затиранию файловых областей.

Беспечность проявляют и работники государственных учреждений разных стран. Так 30 января 2003 года Британское правительство на сайте **www.number-10.gov.uk** опубликовало документ "Iraq — Its Infrastructure Of Concealment, Deception And Intimidation" ("Ирак — инфраструктура утаивания, обмана и устрашения"), который был представлен в формате Microsoft Word и вызвал несколько скандалов.

Было установлено, что большие фрагменты документа являются плагиатом и скопированы из статьи Ибрахима аль-Мараши (Ibrahim al-Marashi) без разрешения автора и даже без ссылки на него, но с сохранением синтаксических ошибок. Однако к вопросам уничтожения данных относилось другое открытие.

В документах Word сохраняется информация о том, кто и когда этот документ редактировал. Так по содержимому документа легко выяснить, кто именно участвовал в его подготовке, и эта информация стала общедоступной, чего не должно было произойти.

Теперь Британское правительство публикует подобные документы в формате PDF, который не предусматривает сохранения информации об авторах вносимых изменений. Но и при использовании формата **PDF** возможны накладки.

Письмо "Вашингтонского снайпера"

В октябре 2002 года в Вашингтоне и окрестностях действовал снайпер, убивавший людей без видимой причины. 24 октября был произведен арест двух подозреваемых, и убийства прекратились.

26 октября 2002 года газета "The Washington Post" выложила на своем сайте отсканированную копию письма "Вашингтонского снайпера", представленную в формате PDF. В письме, среди прочего, описывалась процедура, при помощи которой снайпер собирался получить 10 миллионов долларов. Правительство должно было перечислить деньги на счет платиновой карты Visa.

Письмо содержало все необходимые атрибуты счета и имя женщины, у которой была украдена карта. Также в нем фигурировало 3 телефонных номера. "The Washington Post" отретушировала PDF-документ, скрыв персональную информацию от публичного распространения. В результате, вместо некоторых фрагментов текста появились черные прямоугольники.

Однако метод уничтожения информации был выбран неверно. При выводе на печать все выглядело именно так, как и было задумано: прямоугольники

на месте текста. Но даже при отображении на экран в программе Acrobat Reader (особенно на медленных компьютерах) можно было заметить, что сначала появляется текст, а затем поверх него прорисовываются черные заплатки. То есть персональная информация оказалась просто прикрыта, но не уничтожена.

Действительно, прямоугольники, перекрывающие изображение, легко удалить при помощи инструмента TouchUp Object Tool, входящего в коммерческую программу Adobe Acrobat. И полный текст письма "Вашингтонского снайпера" чуть ли не в тот же день можно было найти в Интернете.

Письмо "Вашингтонского снайпера" оказалось не первым случаем, когда журналисты раскрывали чужие секреты, не позаботившись об их правильном уничтожении. Так в середине апреля 2000 года на сайте **NYTimes.com,** принадлежащем одноименной газете, появилась публикация "Secrets of History: The C.I.A. in Iran" ("Секреты истории: ЦРУ в Иране"), в рамках которой было выложено несколько документов, ранее считавшихся секретными. Документы были отсканированы и сохранены в формате PDF. Публикация вызвала интерес, и в середине июня "The New York Times" выложила в Интернет еще несколько документов в виде PDF-файлов.

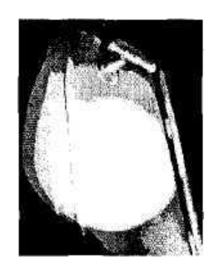
Чтобы не подвергать опасности людей, чьи имена упоминались в документах, некоторые фрагменты оказались закрыты черными прямоугольниками. Но заплатки оказались просто наложены поверх текста и могли быть очень легко удалены. Таким образом, "The New York Times" невольно раскрыла имена нескольких агентов ЦРУ.

Однако, похоже, мало кто учится на чужих ошибках. В конце октября 2003 года в Интернете был опубликован отчет министерства юстиции США под названием "Support for the Department in Conducting an Analysis of Diversity in the Attorney Workforce" ("Помощь министерству в проведении анализа различных работников прокуратуры"). Однако ключевые фрагменты отчета, включая все выводы и рекомендации, были вымараны перед публикацией — правке подверглось более половины страниц.

Но, несмотря на то, что еще в ноябре 2002 года министерство юстиции приобрело корпоративную лицензию на программу Appligent Redax 3.0, предназначенную для полного удаления информации из PDF-документов, Redax не был использован и все фрагменты, скрытые в упомянутом выше отчете, можно без труда восстановить.

Как видно из приведенных примеров, далеко не у всех есть под рукой инструменты, позволяющие качественно выполнять уничтожение информации, а те, кому инструменты доступны, не всегда заботятся об их применении. И происходит это, скорее всего, от элементарного непонимания важности правильного уничтожения данных.

Глава 16



Особенности реализации DRM

С развитием высокоскоростных каналов передачи информации все актуальнее становятся вопросы обеспечения управления цифровыми правами (Digital Rights Management, DRM).

16.1. Что такое DRM

Основная идея DRM заключается в том, чтобы предоставить владельцу авторских прав на некоторое произведение (музыку, книгу, видеофильм) возможность решать, кто и на каких условиях должен получать доступ к этому произведению, и закрепить это решение с помощью технических средств.

Так, например, если владелец прав (издатель) принимает решение, что книгу можно читать с экрана компьютера, но нельзя печатать, то у конечного пользователя, купившего электронную версию книги, не должно быть технической возможности получить бумажную копию.

Кроме таких очевидных вещей, как возможность печати, существуют и более сложные варианты прав. Ту же печать можно ограничивать в терминах "k страниц в n дней", т. е. в течение любых n дней разрешено напечатать не более k страниц.

Также иногда требуется ограничить число прослушиваний музыкальной композиции или число просмотров ознакомительной версии фильма.

Можно ограничивать и период времени, в течение которого разрешен доступ. Причем ограничение может накладываться как сверху (произведение доступно в течение трех дней), так и снизу (доступ будет разрешен только после определенной даты). Второй тип имеет смысл, когда информация доставляется заранее, но ее использование разрешается одновременно во всем мире.

Для таких произведений, как электронные книги, можно задавать, сколько раз можно передавать книгу другому лицу и на каких условиях ее можно сдавать в аренду. Это позволяет организовывать электронные библиотеки,

функционирующие максимально похоже на обычные библиотеки с бумажными книгами.

На базе DRM может строиться и защищенный документооборот, в рамках которого определяются правила использования каждого документа, циркулирующего внутри компании.

Но при любой комбинации прав доступа всегда запрещается такая операция, как получение незащищенной электронной копии произведения. Ведь если такая копия может быть получена, все остальные ограничения перестают работать.

16.2. Возникновение DRM

Происхождение идеи DRM объясняется очень просто.

Когда-то для записи музыки и фильмов использовались только аналоговые форматы: и виниловая пластинка, и обыкновенный кассетный магнитофон, и видеомагнитофон формата VHS. А для аналоговых устройств свойственна потеря качества при перезаписи — и воспроизводящий, и записывающий тракты не идеальны, да и носитель информации (винил или магнитная лента) подвержен различным воздействиям, ухудшающим качество записи. В любом случае, копия всегда получается чуть хуже оригинала, а оригинал со временем изнашивается. И это сильно затрудняет нелегальное тиражирование записей. Правда, в свое время гиганты киноиндустрии пытались запретить продажу бытовых видеомагнитофонов, предрекая, в противном случае, разорение большинства кинокомпаний. Однако по прошествии нескольких десятилетий можно с уверенностью сказать, что обещанных ужасов так и не случилось, хотя видеомагнитофон сейчас есть почти в каждом доме.

С развитием технологий и удешевлением производства сложной электроники цифровые носители информации стали постепенно перебираться из профессиональной сферы в жизнь обыкновенных людей. Одним из первых цифровых носителей, получивших широкое распространение, оказался звуковой компакт-диск. Но бытовые музыкальные центры позволяли только воспроизводить диски, а для записи звука все равно приходилось использовать аналоговый носитель.

Когда появились бытовые цифровые аудиомагнитофоны, получить точную цифровую копию все равно было невозможно. Музыка продавалась на компакт-дисках, записанная с частотой дискретизации 44,1 КГц, а частота дискретизации цифровых магнитофонов сознательно была выбрана равной 48 КГц. И при записи с компакт-диска на магнитофон сначала приходилось преобразовывать цифровую информацию в аналоговый сигнал, а потом выполнять обратное преобразование, но уже с другой частотой.

Только когда появились сравнительно недорогие приводы для записи компакт-дисков, идеальная копия, наконец, стала реальностью. Примерно в то же время производительность персональных компьютеров стала достаточной для того, чтобы обрабатывать (по крайней мере, воспроизводить) в реальном времени аудио- и видеоинформацию, обработанную сложными алгоритмами сжатия. И, вдобавок, в распоряжении простых пользователей оказался высокоскоростной (по сравнению с модемом) доступ в Интернет.

Все это вместе привело к тому, что почти любой пользователь мог создать, например, высококачественную цифровую копию музыкального диска и быстро передать ее на другой конец света.

Разумеется, продавцы контента не захотели мириться даже с теоретической возможностью такого распространения информации, защищенной авторскими правами, и стали разрабатывать и внедрять всевозможные средства контроля над использованием продаваемой ими продукции. Законодательные пути привели к законам типа DMCA (Digital Millenium Copyright Act), а усилия в технической области как раз и воплотились в идею DRM.

Одним словом, DRM потребовался для того, чтобы предотвратить возможность свободного создания и распространения незащищенных и неконтролируемых копий различных произведений и документов.

16.3. Очевидное препятствие

При реализации DRM сразу становится очевидной одна проблема. В отличие от задачи обеспечения секретности, где достаточно было зашифровать все данные и сохранить ключ в тайне, для обеспечения DRM методов криптографии уже не хватает.

Разумеется, криптография все равно в той или иной форме применяется для защиты контента. Но если пользователь имеет возможность доступа к содержимому произведения (например может читать книгу или слушать музыку), значит, все ключи шифрования уже присутствуют в системе и для получения незащищенной копии остается только найти эти ключи и с их помощью получить копию документа, не содержащую ограничений.

Следовательно, для обеспечения DRM приходится использовать и методы защиты, не имеющие математического обоснования стойкости. Или, другими словами, система DRM должна всеми доступными средствами препятствовать как пассивному (исследование), так и активному (модификация) вмешательству в ее функционирование.

Если в системе DRM присутствует аппаратная составляющая (как было, например, до тех пор, пока не появились программные проигрыватели DVD),

обеспечение защиты и контроль прав может выполняться вне персонального компьютера. Но как только происходит переход к чисто программным системам, гарантированно защититься от обхода DRM становится невозможно.

Защищать информацию от одного типа доступа, не препятствуя при этом доступу другого типа, в условиях, когда противник имеет полный контроль над процессами, происходящими в вычислительной системе, крайне сложно. Однако разработчики средств защиты с разной степенью успеха пытаются применять самые разнообразные способы для того, чтобы создать инфраструктуру, в которой станет возможно использовать защищенные электронные произведения, никогда полностью не выходящие из-под контроля издателя.

16.4. Защита электронных книг

Устоявшихся и хорошо себя зарекомендовавших DRM-решений, пожалуй, нет ни для одного вида цифровых произведений. Существуют эффективные методы и для обхода шифрования, применяемого для защиты содержимого DVD-дисков, и для создания незащищенных копий музыкальных композиций, распространяемых в формате WMA (Windows Media Audio) с защитой DRM версии 2.

Но рассматривать основные аспекты DRM можно на любом примере, в том числе и на примере электронных книг.

16.4.1. Adobe PDF Merchant (Adobe.WebBuy)

Как уже говорилось, в Acrobat Reader 4.0.5 появилась поддержка модуля защиты, предназначенного для работы с электронными книгами, продаваемыми через Интернет. PDF Merchant был первым модулем защиты для PDF, позволяющим работать с ключами шифрования, длина которых превышает 40 бит.

При попытке открытия защищенного документа модуль защиты выполнял следующие действия. На сервер, отвечающий за управление правами, отсылалась информация о покупке документа, к которому запрашивался доступ, и один или несколько идентификаторов вычислительной среды, таких как ID компьютера или процессора, серийный номер диска или имя пользователя. Сервер проверял, разрешен ли доступ к документу (была ли действительно на него куплена лицензия), и если все было правильно, генерировал и присылал RMF-файл. Скорее всего, RMF является аббревиатурой от Rights Management Format, во всяком случае, RMF-файл представлял собой XML-документ, в котором хранилась такая DRM-информация, как замаскированный ключ шифрования PDF-документа, перечень разрешенных операций (например печать) и сертификат для проверки подлинности лицензии.

В проверке подлинности лицензии были задействованы два 1024-битовых открытых ключа RSA. Один из этих ключей, вероятно, принадлежат издателю и использовался для проверки цифровой подписи лицензии. А второй ключ, скорее всего, принадлежащий Adobe, применялся для заверения подлинности открытого ключа издателя.

В RMF-файле присутствовало не менее одной записи, описывающей проверки, при успешном прохождении которых должен был открываться доступ к документу. Каждая запись содержала тестовое условие (равно, не равно, больше), имя проверяемого компонента (CPU, USERID, UTC), требуемое значение (идентификатор процессора или пользователя или дату, до наступления которой разрешалась работа с документом) и замаскированное значение ключа шифрования документа.

Несколько условий могли быть скомбинированы с помощью логических операций AND И OR. ЕСЛИ все необходимые проверки выполнялись успешно, то замаскированное значение ключа шифрования документа одной из записей превращалось в ключ шифрования, который и использовался для расшифровки содержимого документа.

В целом, шифрование применялось таким образом, что создать RMF-файл без участия Adobe было практически невозможно. Но зато наличие RMF, соответствующего определенному документу, позволяло вычислить ключ шифрования без физического доступа к компьютеру, для которого был лицензирован этот документ, — проверка условий выполнялась логически (возвращалось состояние истина/ложь), но характеристики компьютера явно не участвовали в вычислении ключа шифрования. К тому же, если несколько проверок объединялись оператором AND, КЛЮЧ шифрования можно было извлечь из записи, соответствующей любой из проверок, не обязательно было выполнять их все.

Можно считать, что стойкость DRM, реализуемая в PDF Merchant, основывалась, в основном, на некоторой сложности выполняемых для получения ключа действий. Однако для построения надежного решения DRM этого явно недостаточно.

16.4.2. Adobe DRM (EBX_HANDLER)

Еще одно решение в области DRM для электронных книг, продвигаемое в настоящее время компанией Adobe, как уже упоминалось ранее, было изначально разработано для программы GlassBook Reader.

GlassBook Reader реализовывал управление правами доступа в соответствии со спецификацией протокола обмена электронными книгами (Electronic Book Exchange, EBX), разрабатываемым организацией EBX Workgroup.

Основная идея протокола заключается в том, что при активации программы GlassBook Reader генерируется пара ключей асимметричного криптографического алгоритма и открытый ключ регистрируется на сервере, а секретный сохраняется на компьютере пользователя. В процессе приобретения лицензии на книгу, Reader получает так называемый ваучер — XML-файл, содержащий ключ документа, зашифрованный на открытом ключе пользователя, список прав доступа к документу и вспомогательную информацию для проверки подлинности ваучера.

Таким образом, для получения ключа шифрования документа необходим секретный ключ пользователя. Несмотря на то, что секретный ключ считается принадлежащим пользователю, сам пользователь не имеет к нему доступа — только Reader знает, как этот ключ может быть извлечен.

В GlassBook Reader существовало два пути доступа к секретному ключу пользователя. Один из путей требовал вычисления значения хэш-функции от некоторых параметров компьютера, таких как ID процессора и серийный номер диска. Разумеется, при смене оборудования результат хэш-функции изменялся и доступ к ключу становился невозможен.

Для того чтобы пользователь не потерял доступ ко всем приобретенным книгам при смене оборудования, вероятно, и был предусмотрен второй способ доступа к секретному ключу, никак не завязанный на характеристики компьютера. Используя этот способ, очень легко было вычислить секретный ключ пользователя, а значит, и получить неограниченный доступ ко всем легально приобретенным документам.

Надежность модели DRM, реализуемой GlassBook Reader, основывалась не только на сложности получения доступа к секретному ключу пользователя. Скорее всего, разработчики полагались в некоторой степени на протектор PACE InterLok, который использовался для предотвращения исследования кода GlassBook Reader. Однако InterLok плохо справлялся с возложенной на него задачей, что делало GlassBook Reader уязвимым для целого спектра атак на DRM.

После перехода GlassBook Reader в собственность Adobe и превращения его в Adobe eBook Reader никаких значительных технических улучшений в плане снижения уязвимости системы DRM так и не было сделано.

Однако с появлением Acrobat 6 работа с электронными книгами была перенесена в Adobe Acrobat и Adobe Reader, а Adobe eBook Reader, похоже, перестал развиваться.

16.4.3. Общая проблема с DRM для PDF

С обеспечением DRM в отношении документов в формате PDF существует еще одна сложность.

Дело в том, что на протяжении многих лет защита документа обеспечивалась по следующей схеме:

- Acrobat выяснял, какой модуль был использован для защиты документа, и передавал ему информацию о защищенном документе;
- модуль защиты проверял права доступа, и если пользователь имел все необходимые разрешения для открытия документа, в Acrobat возвращался ключ шифрования;
- Acrobat использовал ключ, полученный от модуля защиты, для расшифрования фрагментов документа перед отображением их на экране.

В этой схеме очень легко найти слабое место. После того как модуль защиты вычислил ключ шифрования, этот ключ передается в Acrobat, а значит, может быть перехвачен и использован внешней программой для получения незащищенной копии документа.

Возможность такой атаки практически сводила к нулю все усилия по построению DRM, т. к. любые ограничения становились бессмысленными, если можно было перехватить ключ шифрования при первом показе документа на экране.

Начиная с шестой версии продуктов семейства Acrobat, модули защиты получили возможность самостоятельно решать, каким именно способом будет зашифрована та или иная часть PDF-документа. И при такой реализации перехват ключа уже не работает, т. к. передачи ключа между отдельными узлами защиты просто не происходит.

Но тут возникает другая проблема. Начиная с шестой версии, показ электронных книг ведется не через eBook Reader, а через Adobe Acrobat или Adobe Reader. И, разумеется, сама программа, показывающая документ, имеет полный доступ к его содержимому, какой бы алгоритм шифрования не использовался в модуле защиты.

Но и Acrobat, и бесплатный Reader поддерживают модули расширения. И совершенно очевидно, что если противник сможет создать свой модуль расширения и сделать так, чтобы он был загружен в тот момент, когда открывается защищенная книга, то из этого модуля (фактически выполняющегося как часть программы просмотра) можно получить полный доступ к содержимому книги.

Разумеется, Adobe предприняла некоторые усилия для того, чтобы предотвратить возможность загрузки посторонних модулей расширения в тот момент, когда идет работа с книгами, защищенными DRM. Но, как было показано в *разд. 2.3*, самые последние версии программ семейства Acrobat продолжают загружать модули расширения, имеющие поддельную цифровую подпись, но только в "несертифицированном" режиме, в котором работа с защищенными электронными книгами невозможна.

Но после того как модуль расширения был загружен и получил управление, он может "убедить" Acrobat в том, что все загруженные модули расширения имеют правильные сертификаты, а значит, можно и открывать электронные книги.

К сожалению, представители Adobe утверждают, что загрузка модулей расширения с поддельными подписями является нарушением лицензии (что, в общем, справедливо) и, следовательно, не может считаться проблемой безопасности. Однако здравый смысл подсказывает, что к вопросам безопасности относится все, что хоть как-то влияет на возможность использовать данные любым способом, отличным от способа, запланированного издателем.

16.4.4. Microsoft LIT

Еще одна независимая попытка построения рынка электронных документов была предпринята корпорацией Microsoft. Были разработаны новый формат электронных книг LIT (Literature) и бесплатная программа Microsoft Reader для просмотра представленных в нем документов.

Формат внутреннего представления данных в LIT основан на спецификации OEB (Open eBook Publication Structure), разрабатываемой организацией Open eBook Forum, к настоящему моменту объединившейся с EBX Workgroup.

При разработке LIT было предложено реализовать 5 уровней защиты документа с возрастанием защищенности при переходе к следующему уровню. Однако уровень 1 (без защиты вообще) и уровень 4 (ограничено защищенный от копирования) были отброшены как бесперспективные. В результате, Microsoft Reader поддерживает документы трех степеней защиты:

- уровень 2 — "опечатанный" (Sealed). Содержимое книги упаковано и зашифровано, чтобы избежать нарушений целостности. При этом книга не считается защищенной от копирования;
- уровень 3 — "надписанный" (Inscribed). То же, что и "опечатанный", но на титульную страницу электронной книги выводится информация о покупателе (как правило, это имя покупателя и уникальный идентификатор покупки). Подобная информация позволяет выяснить происхождение конкретного экземпляра, что является дополнительным стимулом к честному использованию;
- уровень 5 — "для личного пользования" (Owner-Exclusive). Книга зашифрована и может быть открыта для чтения только на том устройстве, которое было активировано для прочтения именно этой книги.

Для использования книг, имеющих пятый уровень защиты (Owner-Exclusive), необходимо активировать Microsoft Reader, привязав его к учет-

ной записи в системе MS Passport. К одному паспорту можно привязать суммарно до 8-ми разных персональных компьютеров и устройств Pocket PC. Тогда книги, купленные с одного из этих устройств, можно будет читать с любого другого компьютера, привязанного к тому же паспорту.

При выполнении активации на компьютер пользователя передаются несколько библиотек, необходимых для работы с защищенными книгами. Эти библиотеки очень неплохо защищены от исследования. Так, например, если один компьютер дважды привязать к одному и тому же паспорту, то библиотеки окажутся совершенно разными, хотя и будут работать одинаково.

Однако как минимум одному британскому программисту Дэну А. Джексону (Dan A. Jackson) удалось разобраться с устройством средств защиты, применяемых в Microsoft Reader, и разработать программу Convert LIT, позволяющую переводить Owner-Exclusive книги в Sealed, а также извлекать все содержимое книг и сохранять его в виде набора файлов OEB. Convert LIT распространяется под лицензией GPL (GNU General Public License) бесплатно и с исходными текстами.

Microsoft не стала устраивать истерии по поводу появления программы для снятия защиты с электронных книг, в результате чего инцидент прошел практически незамеченным средствами массовой информации. С тех пор было выпущено несколько обновлений Microsoft Reader, препятствующих снятию защиты, но автор Convert LIT в ответ оперативно создает новые версии своей программы.

Похоже, что Convert LIT умеет напрямую работать с библиотеками защиты, получаемыми при активации, и может заставить их расшифровать книгу. Однако существует и другой сценарий атаки, позволяющий расшифровывать документы LIT и извлекать их содержимое.

Дело в том, что Microsoft при разработке LIT использовала несколько существующих технологий. Так для вычисления хэш-функции применяется модификация алгоритма SHA1, в которой 9 трансформаций из 80 были изменены. Шифрование выполняется при помощи модифицированной версии алгоритма DES. Но, самое главное, внутреннее устройство файлов LIT очень похоже на внутреннее устройство файлов CHM (Compiled HTML Help file), применяемых для хранения справочной информации.

Но про CHM-файлы достоверно известно, что они являются одной из реализаций структурированного хранилища, т. е. поддерживают интерфейс IStorage. Следовательно, можно сделать вывод, что и LIT-файлы поддерживают этот интерфейс. Если противнику удастся перехватить управление в тот момент, когда все защитные механизмы отработают и Microsoft Reader будет иметь указатель на объект, соответствующий корню хранилища, снятие защиты станет тривиальным. Достаточно будет вызвать методы интерфейсов

IStorage и IStream, описанные в MSDN (Microsoft Developers Network), чтобы перебрать все вложенные хранилища и потоки и сохранить их на диске в виде директорий и файлов.

16.4.5. Тенденции рынка электронных книг

Не совсем понятно, что ждет индустрию электронных книг в будущем. На настоящий момент прибыль от продаж защищенных электронных изданий составляет единицы, если не доли процента от прибылей, получаемых за счет бумажных книг. И особых поводов к изменению такого соотношения пока не заметно.

Маловероятно, что электронные книги плохо продаются из-за того, что кто-то снимает с них защиту и начинает бесплатно раздавать направо и налево. Скорее уж пользователи опасаются иметь дело с защищенными книгами, т. к. многим пришлось столкнуться с проблемами потери доступа из-за сбоев в программном обеспечении или изменении аппаратной конфигурации. Также, несмотря на все удобства, которые сопутствуют электронным книгам (а это гиперссылки, возможности поиска и аннотирования, озвучивание и многое другое), защищенные книги во многом ограничивают пользователя.

Далеко не всегда книгу удается читать там, где хочется. Ведь многие люди используют, например, Linux, но ни Microsoft Reader, ни Adobe eBook Reader под Linux не работают. Да и бумажную книгу, которая больше не нужна, можно подарить или просто дать почитать. А выполнение подобных действий с электронными книгами почти всегда запрещается издателями.

В любом случае, 9 сентября 2003 года онлайновый книжный магазин, принадлежащий Barnes&Noble, одной из крупнейших мировых книготорговых компаний, объявил о прекращении продаж электронных книг в форматах Microsoft Reader и Adobe Reader.

16.5. Digital Property Protection

Компанией Infraworks, занимающейся вопросами борьбы с воровством интеллектуальной собственности, была разработана технология InTether (на привязи), которую сами разработчики предпочитают называть не DRM, а DPP — Digital Property Protection (защита цифровой собственности). Возможно, разница между DPP и DRM — это всего лишь игра терминов, позволяющая не бояться обвинений в нарушении патентов на системы DRM. Во всяком случае, у DRM и DPP очень много общего.

Отличительная особенность InTether заключается в том, что эта технология может применяться для защиты документов практически любого типа — совершенно не обязательно знать, с чем именно будет иметь дело защита.

После установки InTether на компьютер у конечного пользователя появляется возможность получать из Интернета или другого источника защищенные документы и размещать их на диске. На самом деле, в файловую систему будет помешен только файл нулевого размера, а содержимое окажется спрятанным в специальном защищенном хранилище. Но когда загружены драйвера защиты, создается ощущение, что файлы действительно существуют и имеют ненулевой размер.

При попытке открытия защищенного документа любой программой в ход вступает модуль защиты, предупреждающий пользователя о том, какие ограничения наложены на этот документ. Набор поддерживаемых ограничений довольно широк и включает в себя такие возможности, как уничтожение документа через 10 минут после открытия, запрет на сохранение копии документа, запрет на использование буфера обмена и многое другое.

Если пользователь подтверждает свое желание открыть документ, то программа, используемая для этого, попадает как бы в изолированный мир. Для пользователя все будет работать практически как обычно: можно сохранять файл, можно копировать выделенный текст в буфер обмена. Но ни один другой процесс не увидит результатов этих действий, т. к. защита лишь эмулирует их для процесса, открывшего защищенный файл.

По утверждению главного администратора компании Infraworks, защита, на разработку которой ушло более 3-х лет, состоит из 11 слоев, каждый из которых контролирует целостность всех остальных. И если противнику удастся нейтрализовать один из слоев, это обнаружится в другом слое и защищаемая информация будет уничтожена.

Действительно, защита устанавливает около 10 драйверов в ядро операционной системы, что само по себе выглядит устрашающе. Но ведь общая стойкость защиты определяется самым слабым звеном. А слабое звено, в данном случае, не в драйверах, а в самой операционной системе.

Дело в том, что Windows поддерживает огромное количество различных способов для передачи информации от одного процесса к другому, например:

- COM (Component Object Model, модель компонентных объектов);
- Data Copy (сообщение WM_COPYDATA);
- DDE (Dynamic Data Exchange, динамический обмен данными);
- File Mapping (файлы, отображаемые в память);
- Mailslots (почтовые ящики);
- Pipes (каналы);
- RPC (Remote Procedure Call, удаленный вызов процедур).

И, используя любой из этих методов, программа, открывшая защищенный файл, может передать его содержимое другому процессу, на функционирование которого не будет наложено никаких ограничений.

Разработчики InTether были поставлены в известность относительно найденной уязвимости, и в последней версии программы используется понятие "trusted applications" — приложения, которым разрешено открывать защищенные документы.

Но, скорее всего, проблема осталась, ведь, например, Microsoft Word должен оказаться в списке "trusted applications" для DOC-файлов. Но Word позволяет выполнять программы, написанные на VBA (Visual Basic for Applications). А средствами VBA можно читать и записывать файлы, обращаться к COM-объектам, вызывать функции из динамически загружаемых библиотек и делать многое другое. Следовательно, с большой вероятностью отыщется и простой в реализации способ использования одного из механизмов межпроцессного взаимодействия (Interprocess communication, 1PC).

Хорошо было бы запретить выполнение всех видов в IPC в приложении, открывшем защищенный документ, но такой метод вряд ли позволит достичь желаемого результата. Ведь многие современные программы активно используют средства IPC, и их блокировка, скорее всего, приведет к потере работоспособности.



Глава 17

Стеганографическая защита данных

Как было рассказано в предыдущих главах, криптографии, опирающейся на методы, имеющие математическое обоснование стойкости, обычно достаточно для обеспечения секретности, но уже не хватает для обеспечения DRM. А еще существуют ситуации, в которых оказываются неприменимы даже методы защиты, основанные на принципе "черного ящика". И в таких ситуациях на помощь может прийти стеганография.

Стеганография позволяет скрыть сам факт передачи сообщения. Для этого используется так называемый *стеганографический контейнер,* в котором передаваемое сообщение размещается таким образом, чтобы его было очень трудно извлечь или разрушить.

В качестве стеганографического контейнера может выступать почти все что угодно: газетная заметка, точка в конце предложения, картинка и даже лист белой бумаги. Главное — чтобы существовал способ незаметно разместить в этом контейнере некоторый объем информации.

Не останавливаясь подробно на истории применения стеганографии в разные эпохи, трудно удержаться от упоминания нескольких исторических фактов. Так, во время второй мировой войны в США предпринимались серьезные меры по предотвращению утечки информации за рубеж. Были запрещены к международной почтовой пересылке шахматные партии, детские рисунки и инструкции по вязанию. Также не допускались международные телеграммы, в которых речь шла про заказ и доставку цветов. Существовал даже специальный фонд бумаги, откуда брались чистые листы, чтобы заменить листы, отправляемые жителями США родственникам, проживающим в странах, где с бумагой были проблемы. Все это делалось для того, чтобы помешать передаче скрытых сообщений.

В информационном мире стеганография также получила развитие. Были разработаны методы, позволяющие использовать в качестве стеганографического контейнера многие популярные форматы данных. Лучше всего для этих целей подходят звуковые файлы и графические изображения,

т. к. правильно внесенные искажения не обнаруживаются визуально или слух вследствие особенностей строения органов чувств. Исследования в области цифровой стеганографии продолжаются до сих пор.

17.1. Защита программ в исходных текстах

Иногда коммерческое программное обеспечение распространяется в исходных текстах. И очевидно, что разработчика не устроит ситуация, когда легитимный пользователь сможет выложить исходные тексты купленных программ или модулей в свободный доступ, нанеся тем самым весьма ощутимый ущерб, и остаться при этом неузнанным. Но обычные методы защиты оказываются бессильны в подобных ситуациях. Ведь исходные тексты не выполняют никаких действий и не могут сами по себе известить производителя о нарушении лицензии или попросить пользователя ввести регистрационный код.

На конференции ISDEF 2003 (Independent Software Developers Forum) представителем компании FastReport, Inc. был сделан доклад о разработанной и успешно применяемой системе защиты программ, распространяемых в исходных кодах, методами стеганографии.

Основная идея защиты заключается в том, что каждому покупателю передается уникальный набор исходных текстов, но скомпилированные из любого такого набора программы работают совершенно одинаково.

Технически детали реализации в докладе не раскрывались, но очевидно, что в качестве криптографического контейнера выступают сами файлы с текстами программ. Скорее всего, защита выполняется следующим образом. Перед отправкой исходных текстов покупателю в них с помощью специального стеганографического алгоритма добавляется некоторый идентификатор, связанный с личностью пользователя. Если один из файлов окажется в свободном доступе в Интернете, то с помощью обратного алгоритма разработчики смогут извлечь идентификатор пользователя, а значит, определить и наказать виновного.

По утверждению докладчика, для размещения идентификатора без изменения функциональности программы применяется более 10 различных приемов. Можно предположить, что эти приемы довольно сильно пересекаются со следующим списком:

- изменение регистра букв (для языков, не различающих прописные и строчные буквы, например Delphi);
- изменение локальных идентификаторов;

- изменение порядка следования функций;
- взаимная замена пробелов и символов табуляции;
- изменение стиля отступов для блоков кода (begin/end, {/});
- изменение стиля расстановки пробелов до и после скобок;
- вставка пробелов в конце строк;
- вставка пустых строк;
- изменение порядка операторов case внутри switch.

Представитель FastReport в своем докладе отметил, что после введения подобной защиты и лишения поддержки нескольких пользователей, уличенных в нарушении лицензии на использование исходных текстов, новые версии программ перестали появляться в открытом доступе. Также было сказано, что переформатирование исходного текста не приводит к полному разрушению идентификатора, т. е. стеганографическая вставка обладает достаточно высокой живучестью.

Идея защиты исходных текстов при помощи стеганографии кажется весьма привлекательной, но существует несколько возможных атак, против которых такая защита вряд ли устоит.

Например, если в распоряжении противника окажется два и более набора исходных текстов, подготовленных для разных пользователей, ему будет довольно легко выявить большинство приемов, применяемых для внедрения идентификатора, а значит, и разработать методы нейтрализации этих приемов.

И, разумеется, если противник получит доступ к самой системе, отвечающей за вставку и извлечение идентификаторов, он сможет не только разобраться во всех деталях реализации защиты, но и, возможно, научиться использовать ее в своих целях. Например, для того чтобы создать и выложить в Интернете набор исходных файлов, якобы выданный другому лицу, тем самым спровоцировав разработчиков на жесткие ответные меры.

Следовательно, доступ к средствам защиты должен быть максимально ограничен, но это создает серьезные препятствия на пути использования данной технологии.

17.2. Защита от утечек информации

В крупных организациях, имеющих большое число сотрудников, почти всегда найдется некоторая информация, которая не должна выходить за пределы фирмы. Но защита от утечек информации представляет собой серьезную техническую проблему.

Пока к информации имеет доступ только один человек, выявить источник утечки, как правило, не составляет труда.

Если к разглашенной информации имел доступ небольшой круг лиц, все они сразу же окажутся под подозрением и их действия станут контролироваться с особой тщательностью.

Но если доступ был разрешен всем сотрудникам, то подозревать их всех — занятие бессмысленное. Лучше использовать методы стеганографии.

Общая идея та же самая, что и при защите исходных текстов, — в каждый документ необходимо внедрить некоторый скрытый идентификатор, который позже может быть использован для обнаружения утечки.

Почти все используемые в настоящее время офисные форматы (а именно они преимущественно используются в деловом документообороте) позволяют легко добавлять информацию, которая не повлияет на представление или печать документов. Но обеспечить сохранность и невозможность обнаружения подобного идентификатора очень и очень сложно.

Идеальным решением стала бы разработка собственного формата, в котором было бы легко скрывать небольшие порции информации, и программы, для работы с этим форматом. Но обеспечение всей необходимой для корпоративного документооборота функциональности требует огромных затрат, и создать подобный продукт вряд ли окажется под силу большинству компаний.

17.3. Альтернатива DRM

Использование DRM — далеко не единственный технический способ защиты интересов правообладателей. Возможны и другие решения.

Часто пользователи негативно относятся ко всем видам DRM, т. к. не хотят мириться с ограничениями, накладываемыми на порядок использования защищенных произведений. В вопросах, касающихся авторских прав, существует понятие Fair Use — добросовестное использование.

В рамках Fair Use любой человек может, например, совершенно легально цитировать произведения, находящиеся под защитой законов об авторском праве. Но многие системы DRM лишают пользователей почти всех возможностей, положенных им по Fair Use. Следовательно, для того чтобы реализовать предусмотренное законом право на добросовестное использование какого-либо произведения, приходится деактивировать DRM и снимать защиту. А отсюда недалеко и до снятия защиты со всех документов без разбора.

Но если правообладатель получит уверенность, что документы не станут бесплатно распространяться, можно будет предоставить пользователям доступ и к незащищенным документам.

Весьма элегантное решение было предложено и применялось компанией Peanut Press, продававшей электронные книги для органайзеров Palm. На титульном листе каждой книги фигурировал номер кредитной карты, использованной при покупке. Человек, купивший книгу, мог без труда передать ее другому лицу, но вряд ли нашлось бы много желающих — номер кредитной карты лучше держать в секрете.

Если отказаться от таких экстремальных мер, как номер кредитки на обложке, то неплохих результатов можно достичь и с помощью стеганографии.

К каждому продаваемому экземпляру произведения необходимо добавлять некоторый незаметный водяной знак, позволяющий идентифицировать покупателя. Если конкретный экземпляр окажется в свободном доступе, проанализировав водяной знак, правоохранительным органам гораздо легче будет найти и наказать виновного. Кроме того, применение стеганографии, в отличие от DRM, хорошо тем, что если с документа защита снята полностью, в этом очень легко убедиться. Но абсолютной уверенности в том, что водяной знак полностью удален из произведения, получить практически невозможно. А в таких условиях распространение документов после удаления водяных знаков становится весьма опасным — гораздо опаснее, чем распространение документов со снятой защитой.

Глава 18



Причины ослабления средств защиты

Разработать по-настоящему надежное программное средство защиты, хорошо справляющееся со всеми возложенными на него задачами, довольно сложно. Многие компании берутся за подобные проекты. Некоторые из них даже достигают коммерческих успехов. Но действительно хорошо справляются единицы.

Рассмотрим несколько основных причин, приводящих к неудачам в попытках разработки средств зашиты.

18.1. Непрофессионализм

В любой области деятельности лучше поручать важную работу профессионалам. Они способны быстро принимать правильные решения в сложных ситуациях. Для этого им приходится долго учиться и практиковаться под присмотром более опытных коллег, которые должны быть готовы прийти на помощь в случае ошибки. Но почему-то при разработке средств защиты обычно все не так.

18.1.1. Иллюзия простоты

Когда над каким-либо проектом работает группа программистов и выясняется, что в существующей системе необходимо предусмотреть функции защиты, чаще всего руководитель проекта возлагает реализацию защиты на плечи одного из имеющихся исполнителей.

Или представим ситуацию, когда программист-одиночка разрабатывает программу, основная функциональность которой относится к той области, в которой он является высококлассным специалистом. И вот, появляется идея защитить часть данных при помощи шифрования. Неужели человек станет тратить время на поиск специалиста по безопасности, а потом изыскивать деньги на оплату его услуг? Скорее всего, нет.

Стоит ли привлекать новых людей, если существующие программисты прекрасно знают свою работу? Надо ли тратить деньги на оплату услуг специалиста по безопасности? Ведь в защите информации нет ничего сложного! С помощью функции из трех строк можно перевести пароль в такой вид, что никто никогда не сможет отгадать, что это за пароль. А еще три строки позволят зашифровать данные. И вот защита готова и даже работает. Ведь если ввести неправильный пароль, то программа ничего не покажет. И при просмотре файла с защищенной информацией в шестнадцатеричном редакторе не видно ни пароля, ни самих данных.

При таком подходе к разработке защиты обычно удается сэкономить некоторое количество денег и времени. Правда потом, когда защита будет взломана, за построение нормальной защиты (если разработчики дорожат своей репутацией) все равно придется заплатить.

Может получиться и по-другому. Иногда человек, перед которым ставят задачу реализовать, например, цифровую подпись, обращается к поисковой системе Google, находит криптографическую библиотеку с подходящей лицензией и использует реализованный в ней механизм подписи "как есть". С чувством хорошо выполненного долга он сдает работу и очень сильно удивляется, когда даже стойкий криптографический алгоритм не смог предотвратить взлом.

Защита информации — это такая область, где нельзя быть специалистом частично. Если в других отраслях возможность применения 90 % существующих технологий приводит к результату, который на 10 % хуже идеального, то в защите информации проценты не играют никакой роли. Полученное решение будет или защищенным, или незащищенным.

Но, к сожалению, понимание того, что специалисты по защите информации не зря едят свой хлеб, ко многим приходит даже не с первого раза. Иначе чем можно объяснить многократные попытки улучшения защиты, конечным результатом которых является легко уязвимая система?

18.1.2. Излишнее усердие

Частным случаем непрофессионализма можно считать попытки защищать даже то, что не нуждается в защите.

Иногда разработчики зашифровывают часть несекретной информации точно таким же образом, как и информацию, которая нуждается в защите. Если при этом программа позволяет просматривать расшифрованную несекретную информацию, то иногда удается подменить зашифрованные данные и увидеть их в расшифрованном виде.

Шифрование паролей в ReGet Deluxe

Программа управления выкачиванием файлов ReGet Deluxe позволяет запоминать настройки закачки раздельно для каждого файла и каждого сайта. Если для доступа к серверу, например, по протоколу FTP (File Transfer Protocol) требуется пароль, он тоже сохраняется.

Все настройки хранятся в файле с расширением wrj, который сам ReGet называет файлом очереди (ReGet Junior/Deluxe Queue File). Этот файл представляет собой XML-документ, в котором визуально очень легко найти всю информацию, относящуюся к конкретному файлу или сайту. Но информация об имени пользователя и пароле доступа хранится в зашифрованном виде. Правда, запустив ReGet, можно в свойствах файла или сайта увидеть имя пользователя, а вот пароль будет представлен в виде набора звездочек.

Разумеется, существует возможность выяснить, каким образом выполняется расшифрование, т. к. ReGet способен расшифровывать пароль, не задавая пользователю дополнительных вопросов. Правда, для этого придется исследовать сам ReGet, а это требует довольно высокой квалификации и обычному пользователю недоступно.

Однако на практике пароль можно извлечь, используя только текстовый редактор и сам ReGet. Если в WRJ-файле поменять местами значения атрибутов "Username" и "Password" для нужного сайта, то ReGet в поле, соответствующем имени пользователя, покажет открытым текстом пароль, а имя пользователя окажется представленным в виде звездочек.

Но если бы имя пользователя не шифровалось вообще (что не снижает защищенность, ведь имя все равно показывается в настройках), то заставить ReGet расшифровать пароль и показать его на экране было бы не так просто.

18.2. Влияние законодательства

Как известно, в США долгое время существовали ограничения на размер ключа шифрования, используемого в программах, экспортируемых из страны. Несмотря на то, что эти ограничения уже давно не действуют, последствия воздействия старых правил до сих пор очень сильны и, похоже, будут ощущаться еще довольно долго.

Огромное количество популярных продуктов, так или иначе использовавших шифрование, были вынуждены применять ключи не длиннее 40 бит. За время существования ограничений было создано огромное количество документов с такой защитой, и многие из них продолжают использоваться до сих пор.

С появлением новых версий программ, поддерживающих более длинные ключи, далеко не все пользователи стали использовать новые настройки. Ведь обновление программного обеспечения не происходит одновременно у всех пользователей. Значит, если всегда создавать документы с самыми новыми настройками защиты, некоторые пользователи, все еще использующие предыдущие версии программ, не смогут работать с такими документами. А период времени, по истечении которого у подавляющего большинства пользователей будет установлена версия программы, поддерживающая длинные ключи, иногда исчисляется годами.

В некоторых странах, например во Франции, были свои ограничения, отмененные позже, чем в США, и документы, созданные версиями программ, которые были предназначены для таких стран, гораздо дольше не защищались длинными ключами.

Ну и, разумеется, далеко не все разработчики после снятия экспортных ограничений (окончательно произошедшего в октябре 2000 года) оперативно выпустили обновленные версии своих продуктов.

Как следствие всего перечисленного выше, сейчас (на конец 2003 года) документы, защищенные 40-битовыми ключами, встречаются гораздо чаще, чем документы, при зашифровании которых использовались более длинные ключи.

Если рассматривать все форматы представления данных, поддерживающих защиту шифрованием, то грубо можно выделить три уровня защиты (по максимальному времени, затрачиваемому на получение расшифрованных данных).

1. Моментальные — пользователю придется ждать менее минуты.
2. Гарантированные — ключ может быть найден на современной технике за время, соизмеримое со временем жизни человека.
3. Стойкие — невозможно дать гарантию, что данные когда-либо удастся расшифровать.

Учитывая развитие вычислительной техники, некоторые зашиты со временем могут переходить в более низкий (по номеру и стойкости) уровень.

На настоящий момент к первому уровню можно отнести защиты, в которых применяются нестойкие алгоритмы, а также алгоритмы с ключами длиной до 32 бит. Алгоритмы второго уровня работают с ключами, длина которых лежит в диапазоне от 32 до 64 бит, хотя даже 56-битовый ключ на персональном компьютере будет подбираться очень долго.

Несколько лет назад соотношение этих трех уровней было примерно следующим. Основная масса защит попадала на первый уровень. Правильно реализованные защиты (использовавшие стойкое шифрование с 40-битовьш

ключом) попадали на третий уровень, и совсем уж редкие экземпляры оказывались между ними — на втором уровне.

Сейчас основная масса защит с третьего уровня перешла на второй, и именно он теперь является самым наполненным. Зато появились и новые представители третьего уровня, использующие 128-битовые ключи. И им переход на второй уровень в обозримом будущем не грозит.

Ну а общие тенденции таковы, что в скором будущем основная масса защит, которые сейчас находятся на втором уровне, будет обновлена и прочно обоснуется на третьем уровне. Второй уровень практически исчезнет, а на первом будут либо новые защиты, авторы которых еще ничего не знают о криптографии, либо очень старые защиты, которые больше не обновляются, но продолжают использоваться.

18.3. Претензии на универсальность

Идея создания универсальных средств DRM, пригодных для защиты документов любых типов, кажется очень соблазнительной, но ее практическая реализация почти всегда обречена на неудачу.

Так как DRM для защиты не в состоянии обойтись исключительно методами, имеющими математическое обоснование стойкости, необходимо максимально сократить пространство, в котором защищенные данные присутствуют в расшифрованном виде.

Это сравнительно просто реализовать, если защита встроена непосредственно в программу, воспроизводящую защищенное произведение. Тогда сразу после проверки прав доступа и расшифрования контент может быть отображен на экране (или выведен в звуковой тракт) и уничтожен.

Но если предполагается, что защищенный файл может обрабатываться любой программой и эта программа даже не будет подозревать, что работает в каких-то особых условиях, путь, проходимый расшифрованной информацией, становится чрезмерно длинным. На этом пути существует огромное количество мест, из которых расшифрованная информация может быть похищена.

Возможно, в будущем и будет создана система DRM общего назначения, в которой расшифрованная информация на всем пути до ушей и глаз пользователя окажется надежно защищена от перехвата. Но маловероятно, что эта система будет построена на базе Windows — уж слишком разнообразны и сложны происходящие у нее внутри процессы, чтобы их все строго контролировать. Скорее уж корпорация Microsoft воспользуется патентом №6330670, выданным ей 11 декабря 2001 года, и реализует на практике описанную в нем операционную систему со встроенным управлением цифровыми правами (Digital Rights Management Operating System, DRMOS).

18.4. Погоня за прибылью

Наверное, большинство разрабатываемого программного обеспечения — коммерческое, т. е. создается с целью получения прибыли. И надо всегда помнить, что хотя и принято считать, что разработчик трудится на благо пользователей и в своих действиях руководствуется их интересами, на самом деле это не так.

В интересах разработчика — создать программу, которую будут покупать. В интересах пользователя — получить программу, которая выполняет требуемые функции. Если пользователь увидит, что программа работает не так, как ему хочется, он просто не станет покупать такую программу. Это стимулирует разработчика выслушивать и исполнять желания пользователей.

В системах, связанных с защитой, эта схема работает плохо. Так как недостатки защиты вскрываются лишь тогда, когда защита будет взломана, у пользователя нет возможности отказаться от покупки по причине очевидной неработоспособности программы. Следовательно, у разработчика нет причины изначально создавать надежную защиту.

Вместо этого разработчики предпочитают тратить ресурсы на то, что помогает привлечь пользователей и заставить их покупать программу: на рекламу, добавление полупрозрачных окон и озвученных меню. И практика показывает, что хорошо разрекламированное и раскрашенное, но небезопасное средство защиты продается гораздо лучше, чем качественно выполненная защита с простым интерфейсом.

Также достаточно сильным стимулом к выпуску новых версий программного обеспечения является желание как можно быстрее заполнить новую рыночную нишу. И, следуя этому желанию, на рынок часто выбрасывается сырой программный продукт, в котором вопросы защиты не проработаны до конца — все равно проблемы некоторое время останутся незамеченными. Разработчик зарабатывает желанные деньги, а за его ошибки, в конечном счете, будут расплачиваться пользователи.

18.5. Технологические причины

Разработка программного обеспечения давно уже превратилась в отдельную индустрию, которая живет по своим законам и использует свои технологии. Эти технологии призваны сократить время разработки, повысить качество готовой продукции и, в конечном итоге, повысить прибыть от продажи программ.

18.5.1. Эффективность разработки

Для обеспечения максимальной производительности труда программистов при разработке средств защиты применяются те же самые идеи организации

процесса производства, что и при разработке обычных программ. Используются готовые библиотеки, очевидные решения, создается эффективный и понятный код и т. д.

Но когда ведется разработка методов защиты, не имеющих математического обоснования стойкости, общепринятых приемов лучше избегать. Чем логичнее и очевиднее ведет себя защита, тем проще противнику будет ее проанализировать и понять. Поэтому защита должна быть максимально запутанной — только в этом случае у нее есть хоть какой-то шанс устоять.

18.5.2. Преемственность

Если первая версия программного продукта оказалась более-менее успешной, то через некоторое время выпускается следующая версия, в которой исправляются обнаруженные ошибки и добавляются новые функции, придуманные разработчиками. И этот процесс может повторяться очень долго, т. к. почти в любой программе есть что исправить или добавить.

Разумеется, при выпуске каждой новой версии разработчики стремятся использовать как можно больше из того, что уже было создано, а не пишут весь код заново. Со временем это часто приводит к тому, что почти любой модуль, входящий в состав программы, представляет собой нагромождение заплаток и надстроек, но вся система в целом является более-менее работоспособной.

Если защита добавляется в уже спроектированную систему, то функции безопасности будут реализованы как надстройки над существующими процессами обработки информации. А это редко дает хорошие результаты.

Прежде всего, если программная система изначально не задумывалась для обработки защищенной информации, то с большой вероятностью никакие модификации и дополнения не позволят нейтрализовать все недостатки, присущие базовой системе — проектирование защищенных систем очень сильно отличается от проектирования обычных программных комплексов.

Кроме того, надстройки почти всегда усложняют систему и, следовательно, затрудняют анализ происходящих в ней процессов. Но убедиться в том, что при разработке защиты не было упущено никаких важных моментов, можно, только проведя всесторонний анализ и тестирование защитных функций. Следовательно, убедиться в стойкости системы защиты будет крайне трудно.

Однако мало кто из разработчиков решается на полное перепроектирование, ведь это потребует громадных затрат времени и денег.

18.6. Отсутствие ответственности

Очень серьезным фактором, способствующим появлению многочисленных программ, не способных правильно выполнять обещанные защитные функции, является отсутствие ответственности разработчика за ущерб, понесенный пользователем в результате применения программы.

Программная индустрия, наверное, единственная в мире позволяет разработчикам избегать ответственности за то, что они плохо выполняют свою работу и не держат данных обещаний.

Сейчас при установке почти любой программы пользователю предоставляется возможность ознакомиться с лицензионным соглашением, в котором перечисляется все, что пользователю запрещено делать с купленной программой. И с большой вероятностью в лицензионном соглашении содержится пункт, согласившись с которым, пользователь не сможет предъявить разработчику никаких претензий. Разумеется, пользователь не обязан соглашаться, но тогда он не сможет установить и использовать программу.

С такими условиями распространения обычных программ еще как-то можно смириться. Но, к сожалению, подобным образом поступают и производители средств защиты. А если разработчик защиты не несет ответственности за надежность своих решений, трудно поверить, что пользовательские данные будут находиться в безопасности.

18.7. Сложность контроля

Современные программные системы имеют весьма сложное внутреннее устройство. Для того чтобы, имея в распоряжении исполняемые файлы, получить определенную информацию о происходящих внутри них процессах, требуется провести весьма сложный и дорогостоящий анализ. Но для простых программ такой анализ обычно и не требуется — по результатам работы довольно просто можно оценить, насколько хорошо реализованы те или иные функции.

Однако с защитой, как обычно, все иначе. Качество защиты невозможно оценить по внешним признакам. Значит, требуется выполнение анализа. Но рядовой пользователь не сможет самостоятельно выполнить такой анализ и оценить его результаты. Следовательно, придется нанимать высококвалифицированного специалиста, способного выполнить подобную работу и, разумеется, оплатить его услуги. А это могут себе позволить далеко не все.

Вот и получается, что контроль реализации защитных функций приобретаемого продукта почти никогда не производится. А вспомнив перечисленные ранее технологические и экономические причины, а также отсутствие ответственности, становится совершенно понятно, почему так часто приходится слышать о взломе той или иной защиты.



Часть V

ЗАМЕТКИ ОБ ИССЛЕДОВАНИЯХ СРЕДСТВ ЗАЩИТЫ

Любому разработчику средств безопасности, стремящемуся создать надежный фрагмент защиты, просто необходимо знать, каким образом попытается действовать противник. Взгляд на защиту глазами нападающего дает возможность лучше видеть слабые места. Поэтому последняя часть книги посвящена вопросам исследования средств защиты и анализа программ.

Глава 19



Кому это нужно

Нетрудно определить, кто нуждается в разработке средств защиты информации. Это те люди и организации, которым есть что защищать, т. е. практически все подряд. Большинство нуждающихся, разумеется, используют технологии защиты, созданные другими, но в некоторых случаях средства информационной безопасности разрабатываются собственными силами.

19.1. Время создавать защиту

За разработку своей защиты имеет смысл браться в трех случаях, когда:

- система защиты разрабатывается как коммерческий проект;
- существующие средства защиты не способны обеспечить необходимую функциональность;
- существующие средства защиты не подходят по соображениям безопасности.

Первый случай, когда защита разрабатывается с целью извлечения прибыли, особого интереса не представляет — это обыкновенный коммерческий проект, в котором обеспечение надежности защиты вполне может не играть никакой роли. Единственная цель разработчика — извлечь максимум прибыли.

Во втором случае пользователю необходимо защищать информацию в некоторых уникальных условиях, для которых ни одна из существующих систем не была предназначена. Подобные ситуации появляются регулярно как прямое следствие развития технологий. Пока не было дисков, пригодных для записи фильмов с хорошим качеством изображения, не стоял вопрос и об их защите. Пока не получили широкое распространение мобильные технологии, не требовалось реализовывать стойкие криптографические алгоритмы на процессорах, используемых в телефонах. Разработка новых технологий в любой области — весьма рискованное занятие, но при защите информации риск увеличивается многократно. Опасности подвергаются не только данные, обрабатываемые в период после обнаружения ошибки противником

и до исправления этой ошибки, но и вообще вся информация, защищенная в то время, когда ошибка существовала.

Третий случай занимателен тем, что, несмотря на наличие средств обеспечения безопасности, внешне пригодных для решения поставленных задач, нет никакой уверенности в надежности существующих решений. А если стоимость потери целостности или конфиденциальной информации очень велика (что вполне реально, например, для банковских данных и государственных секретов), имеет смысл затратить ресурсы на разработку собственной реализации защиты. Ведь достичь рациональной уверенности в том, что средства зашиты, созданные кем-то другим, не содержат случайных или намерено внесенных уязвимостей, очень сложно.

19.2. Время исследовать защиту

Для проведения всестороннего исследования средств защиты может быть много причин. Но практически все эти причины основываются на простейших человеческих желаниях: стремлению к власти, деньгам, безопасности, славе, получению удовольствия и восстановлению справедливости. Разберем эти желания подробнее.

19.2.1. Власть и деньги

Если какой-нибудь человек или компания использовали средство защиты информации, а противнику удалось найти в нем уязвимость, противник получает над пользователями определенную власть. Эта власть может выражаться по-разному: как возможность читать засекреченные сообщения, подделывать чужую подпись или, например, шантажировать того, кто использует взломанную систему.

Разумеется, противник подвергает анализу систему защиты, к разработке которой он сам не имеет никакого отношения — основная цель анализа заключается не в проверке стойкости зашиты, а в получении хоть какого-нибудь способа давления на сторону, применяющую уязвимую технологию. Поэтому обычно ищется наиболее простой способ атаки, приносящий плоды за минимальное время и при минимальных материальных затратах.

До начала компьютерной эпохи взломом чужих систем защиты занимались, преимущественно, на государственном уровне, ведь кроме военных и дипломатов мало кто мог себе позволить использовать хоть сколько-нибудь надежные средства защиты.

Сейчас, когда технологии защиты информации вошли в повседневную жизнь и применяются в той или иной форме почти всеми людьми (ввод

PIN-кода или пароля — это уже использование средств зашиты), возможностей для атаки стало гораздо больше. Это обусловлено как существованием самых разнообразных (и далеко не всегда безопасных) средств защиты, так и сравнительно легким обнаружением объектов, которые можно попытаться атаковать — с появлением Интернета защищенные системы и данные можно встретить буквально на каждом шагу.

Если противнику удалось найти слабое место в системе защиты, его последующие действия могут быть самыми разными в зависимости от характера обнаруженной уязвимости.

Пожалуй, самые страшные последствия могут возникнуть, если противник научился читать зашифрованную переписку. При этом он не оказывает активного воздействия, и его присутствие никак не проявляется. Пользователи могут никогда не узнать, что содержимое их сообщений стало известно противнику.

Если противник сможет, например, подделать чужую цифровую подпись, то первый раз подобные действия вполне могут остаться незамеченными. Но если истинный владелец подписи обнаружит, что его подпись стоит на документе, который он видит первый раз в жизни, самым правильным решением будет отозвать ключ подписи, после чего противник теряет все полученное в результате взлома преимущество.

Достаточно распространенный способ использования информации о недостатках в организации безопасности некоторой системы — продемонстрировать уязвимости и предложить помощь в их устранении (разумеется, -за вознаграждение). Однако в подобной ситуации противник сильно рискует — при многочисленных контактах с владельцами взломанной системы довольно сложно сохранять анонимность, а значит, изрядно возрастают шансы быть пойманным и получить обвинения в неправомерном доступе к информации и шантаже.

Нередко встречается и ситуация, когда противник, не имеющий возможности выполнить анализ зашиты самостоятельно (например из-за недостатка технических знаний), за деньги нанимает специалистов, которые и проводят все необходимые исследования.

19.2.2. Самозащита

Как уже упоминалось в начале настоящей главы, в жизни бывают такие ситуации, в которых защищаемая информация стоит очень дорого. И умный владелец информации предпочтет затратить на создание нормальной защиты сумму, соизмеримую с той, в которую оценивается ущерб при потере контроля над защищаемыми данными (при нарушении целостности или

секретности). То есть построение надежной защиты позволит не потерять очень крупную сумму денег.

Однако разработка новой, хорошо выверенной и отлаженной системы защиты не только дорогой, но и весьма длительный процесс. Разработка обычного программного обеспечения — не самый быстротекущий процесс, но программы, связанные с защитой, разрабатывать во много раз сложнее.

Недостаточно сформулировать техническое задание — надо проверить, что никакая комбинация элементов будущей системы не оставит лазейки противнику.

Недостаточно иметь готовую к работе команду программистов — необходимо научить их мыслить в терминах безопасности и писать не простой и эффективный, а надежный и проверяющий все, что только можно, код.

Недостаточно прогнать набор основных тестов — в проверке нуждается каждая ветка каждого алгоритма, во всех возможных режимах.

Поэтому вместо того, чтобы начинать разработку новой системы защиты, иногда бывает проще провести серьезный анализ существующей коммерческой системы на предмет выявления ее пригодности к решению поставленных задач.

Даже если разработка была проведена целиком внутри компании, это еще не означает, что нет никакого смысла в проверке качества реализации. Разработчики и программисты тоже люди, а людям свойственно ошибаться. Поэтому очень полезно иметь еще одну оценку — от людей, не принимавших прямого участия в разработке. Это позволит значительно повысить степень уверенности в стойкости защиты.

Очевидно, что иметь в штате команду специалистов по информационной безопасности, способных быстро и качественно провести анализ защиты, может позволить себе не каждый. Но для проведения исследований можно нанять независимых экспертов, имеющих хорошую репутацию в подобных делах. Как правило, все результаты подобной экспертизы передаются заказчику и не могут быть опубликованы без его согласия.

Подобный анализ "для себя" иногда устраивают и разработчики коммерческих защит, которые хотят получить независимое подтверждение надежности разработанного продукта, которое позже может быть использовано в рекламных целях. Однако "независимость" полученной таким образом оценки весьма сомнительна. Разработчик ожидает от экспертизы положительныгх результатов и, фактически, платит именно за них. Если же экспертное заключение будет негативным, то никто не заставит разработчика его публиковать.

Разумеется, в ходе анализа надежности системы безопасности, которую планируется использовать для защиты очень важных данных, недостаточно

проверять только устойчивость к простейшим видам атак, которые противник сам будет пробовать в первую очередь. Требуется выполнить анализ на максимально доступную глубину. Но начинать можно и с самых простых атак — если окажется, что система неустойчива к ним, остальное проверять не обязательно. Наличие даже одной уязвимости сводит всю защиту к нулю.

19.2.3. Слава

Существует весьма многочисленная категория людей, мечтающих прославиться любым доступным им способом. Этих способов великое множество, и они очень сильно отличаются друг от друга.

Большинство людей, вошедших в историю, на протяжении всей жизни очень много трудились и именно этим снискали себе мировое признание. Так можно начать с разработки собственной версии интерпретатора языка BASIC или эмулятора терминала и через несколько лет стать лидером гигантской софтверной (производящей программное обеспечение) империи или самого крупного открытого программного проекта в истории. Но этот путь не очень надежен — миллионы людей изо дня в день работают не покладая рук, а слава приходит лишь к единицам.

Некоторым людям удается воспользоваться счастливым стечением обстоятельств и прославиться за один день, не прилагая к этому сверхчеловеческих усилий. Достаточно просто оказаться в нужное время в нужном месте, и известность придет сама. Однако фортуна — штука непредсказуемая, и такой способ вхождения в историю тоже не может считаться надежным.

Однако существуют и более стабильные способы заявить о себе. Вспомним хотя бы путь, избранный в 356 году до н. э. Геростратом. Он сжег храм Артемиды Эфесской, одно из семи чудес света, и, тем самым, оставил свой след в истории. Не доходя до таких крайностей, как уничтожение мировых шедевров, вполне можно стать известным, если обратить на себя внимание достаточного числа людей. А в эпоху информационных технологий это не очень сложно.

Сейчас компьютеры используются повсеместно, и от правильной их работы зависит поведение очень многих окружающих нас вещей: от лифтов и светофоров до самолетов и стратегического вооружения. И нарушение в работе компьютеров может привести к серьезным последствиям — достаточно вспомнить все, чем пугали простых людей компании, вовлеченные в решение "проблемы 2000 года". Многие из их прогнозов были не чем иным, как хорошим коммерческим ходом, подталкивающим потратить деньги на решение этой "проблемы". Но ведь существовала и реальная опасность неуправляемых негативных последствий.

То же самое касается и защиты информации. Несмотря на то, что большинство людей, пользующихся компьютерами, не осознает в полной мере всех аспектов информационной безопасности, подсознательно они понимают, что наличие дыр в используемом программном обеспечении может коснуться и их. Уж слишком часто за последнее время пользователям приходится слышать о том, как очередной вирус беспрепятственно шествует по планете, нанося ущерб, оцениваемый миллионами долларов. Ведь из каждого сообщения об обнаружении очередной уязвимости средства массовой информации стараются сделать громкое событие.

Вот и получается, что человеку или компании, обнаружившей слабость в широко распространенном средстве защиты, достаточно оставить сообщение о своем открытии в Интернете. Это сообщение в считанные часы окажется разнесенным по всему миру, появится на тысячах сайтов, попадет через почтовые рассылки миллионам пользователей, а то и выплеснется в прессе и на телеэкранах. И если автор сообщения не забудет указать свое имя, у него есть много шансов остаться в памяти людей, а значит, и в истории.

Однако у жажды славы есть и другие проявления. В компьютерном мире существует довольно многочисленное сообщество людей, занимающихся взломом коммерческого программного обеспечения. Очень часто эти люди объединяются в группы, выбирают себе псевдонимы, а группам — названия, и начинают соревноваться с другими подобными командами.

Цель соревнования, как правило, — это выложить в свободный доступ взломанную новую версию программного продукта раньше, чем это сделает другая группа. Взломом считается то, что программа может использоваться без регистрации или генератор регистрационных кодов прилагается к программе.

Разумеется, члены группы не оставляют своих реальных имен или адресов, и фактически всю известность они создают и закрепляют исключительно в рамках своего сообщества.

19.2.4. Удовольствие

Но кроме людей, рвущихся к славе, власти или деньгам, есть и просто любопытные. В большинстве своем они интересуются не результатом, а процессом исследования. Для них не важно, удастся ли обнаружить уязвимость. Ведь сам процесс исследования может доставлять огромное удовольствие.

Именно из таких любопытных, стремящихся проникнуть в самую суть, и получаются очень хорошие ученые. А научное исследование средств защиты позволяет собирать информацию, необходимую для всестороннего рассмотрения совокупности применяемых технологий безопасности, выявления их

достоинств и недостатков, а также выработки рекомендаций по построению более надежных систем.

Однако ученые в своей работе не используют такую категорию, как мораль. Они решают поставленную задачу любыми доступными средствами. И если перед ученым стоит проблема накормить население страны, то правильным решением будет являться как увеличение производства продовольствия, так и сокращение размера населения.

Не удается однозначно трактовать и последствия научных исследований средств защиты. Результаты анализа могут быть использованы как для улучшения надежности защиты, так и для неправомерного доступа к информации, защищенной другими. Но ученому, по большому счету, это безразлично — главное, что он хорошо выполнил свою работу.

Возможно, именно это является первопричиной идущих довольно давно споров о правильной политике разглашения информации об обнаруженных недостатках средств защиты.

Формально причиной спора является то, что если открыто опубликовать информацию о найденной дыре, то с некоторой вероятностью уязвимость будет использована для нанесения ущерба одному или нескольким пользователям. Следовательно, открыто разглашать информацию о проблемах, связанных с безопасностью, до того, как разработчик уязвимой системы разработает заплатку, ни в коем случае нельзя. И подобной позиции придерживается большинство крупных производителей программного обеспечения.

Но, с другой стороны, если пользователи не будут знать о потенциальной угрозе, а противник ухитрится получить доступ к информации об уязвимости или найдет дыру самостоятельно, последствия атаки могут оказаться более серьезными. К тому же ждать, пока будет выпущена заплатка, иногда приходится очень долго. Ведь далеко не все производители программного обеспечения ставят вопросы безопасности на первое место (хотя в последнее время многие из них заявляют обратное, видимо, стремясь восстановить пошатнувшееся доверие пользователей). Все это приводит к мысли, что только полное описание всех деталей новой уязвимости подготовит пользователей к возможной атаке и подтолкнет разработчиков к скорейшему созданию заплаток. И эту позицию поддерживают очень многие специалисты по информационной безопасности.

19.2.5. Справедливость

Иногда исследование или даже взлом защиты производится с целью обеспечения безопасности государства или восстановления законности. Например,

если информация на компьютере лица, подозреваемого в совершении преступления, полностью или частично зашифрована, то суд может выдать разрешение на анализ использованного средства защиты и на извлечение скрытых данных.

Очень похожая ситуация и с исследованием вредоносных программ: вирусов, троянских коней и т. п. Разработчики антивирусных программ проводят анализ внутреннего устройства вирусов для разработки эффективных методов противодействия им. Это не является исследованием защиты в чистом виде, но, безусловно, относится к информационной безопасности самым прямым образом.

19.3. Синтез и анализ

Как видно, существует довольно много причин заниматься исследованием средств безопасности. Но проводить анализ защиты способен далеко не каждый человек.

У разных людей могут быть совершенно разные склонности. Кому-то ближе естественные науки, общение с природой. Кто-то предпочитает заниматься социальными вопросами, изучать человека и общество. Кто-то тяготеет к точным наукам. Вариантов очень много.

Но если рассматривать людей любой профессии, то их можно разделить на тех, у кого есть способности к программированию, и тех, у кого таких способностей нет. А что отличает программиста от непрограммиста? Какими специальными способностями должен обладать человек, чтобы ему хорошо давалось программирование?

Программист занимается тем, что реализует алгоритмы на заданном языке программирования. В его распоряжении есть набор кубиков — разрешенных языком конструкций, и он складывает эти кубики таким образом, чтобы получилась нужная картинка, представляющая собой алгоритм. А критерий правильности картинки заключается в том, что для любых входных данных можно пройти до цепочке от первого до последнего кубика и получить заданный результат.

Разумеется, почти всегда существует более одного способа сложить кубики, но и конечные картинки будут отличаться по многим характеристикам. Будут расхождения в количестве использованных кубиков (в размере кода) и длине цепочки от первого до последнего кубика (в быстродействии). Некоторые кубики могут стоять неправильно (ошибки). И конечно, в результате будет различаться время, которое программист потратит на складывание картинки из кубиков.

Программистом обычно становится тот, кому быстро удается складывать кубики. А хороший программист делает это быстро, использует минимальное число кубиков и не допускает при этом ошибок. Программирование — это умение решать задачи синтеза.

Исследователем тоже может быть не каждый. Исследования — это решение задач анализа. Когда аналитику показывают целую картинку, он должен уметь быстро разобрать ее на отдельные кубики. Чаще всего, картинка представляется приблизительно — мелкие детали не видны, а некоторых фрагментов просто не хватает. И аналитик додумывает, что должно быть в отсутствующих местах и как именно был реализован тот или иной узел. И, конечно же, пытается найти ошибки, позволяющие заставить картинку вести себя не так, как планировал разработчик.

Почти всегда для того, чтобы понять, как работает программа, требуется начать думать так, как думал программист. И ирония заключается в том, что если программист был хороший и использовал оптимальные решения, то эти решения очень легко предсказать, и тогда даже по очень грубой схеме аналитик будет в состоянии воссоздать все детали. А вот если программу писал человек, не заботившийся о применении наиболее эффективных решений (по незнанию или намеренно), анализ становится значительно сложнее — приходится внимательно разглядывать каждую деталь. Именно по этому при реализации средств защиты требуется уходить от эффективности программирования — это затруднит проведение анализа противником.

Кстати, несмотря на то, что исследователь программ должен хорошо представлять себе, как работает программист, совершенно не обязательно, что человеку, склонному к анализу, будет хорошо даваться синтез. Умение хорошо программировать не является обязательным для аналитика.

Точно оценить соотношение людей, способных к синтезу (программистов), и людей, способных к анализу (исследователей), довольно сложно, но можно предположить, что на 95 программистов приходится примерно 5 исследователей. Подобное соотношение неплохо вписывается в существующую индустрию разработки программного обеспечения. Труд программиста требуется очень часто, а значит, у программиста есть шанс найти работу в огромном числе мест. Но хороших исследователей мало, и их переизбытка почти никогда не бывает, а значит, у аналитика меньше шансов потерять работу.



Глава 20

Интернет — кладезь информации

Когда перед аналитиком ставится задача провести исследование определенного программного комплекса, необходимо с чего-то начать. Разумеется, можно сразу вооружиться отладчиком и дизассемблером и попытаться вникнуть во все детали, но для современных программ такой подход малопригоден — уж слишком велик их объем.

Однако очень много полезной информации об исследуемой программе может быть найдено в Интернете. Главное — знать, что, где и как искать.

20.1. Что искать в Интернете

Как правило, любая система защиты до того, как становится достаточно популярной, чтобы привлечь интерес исследователей, проходит весьма длинный путь развития. И очевидно, что большинство продуктов при переходе к следующей версии изменяется в сторону усложнения. Ведь сначала разработчики создают защиту в таком виде, в котором они считают ее работоспособной. Но, столкнувшись с некоторыми типами атак, которые не были предусмотрены в существующей версии, программисты включают в код новые элементы защиты.

Следовательно, можно предположить, что провести исследование более ранних версий той же системы, а потом перенести полученные знания на актуальную версию будет проще, чем сразу браться за самую свежую версию программы. А значит, необходимо найти предыдущие версии, и чем больше, тем лучше.

Иногда разработчики защиты, сами того не понимая, публикуют информацию, помогающую противнику найти уязвимость. Поэтому очень важно внимательно изучить всю доступную документацию и информацию, приводимую разработчиком в Интернете.

Но нередки случаи, когда, вовремя опомнившись, разработчик удаляет опасные факты из всех описаний. Поэтому желательно просматривать не

Именно с помощью The Wayback Machine иногда удается получить информацию, которая сначала была опубликована разработчиками какой-нибудь программы, а затем удалена. Ведь Wayback Machine хранит копии 30 миллиардов интернет-страниц, собранных с 1996 года.

Кроме того, старые версии страничек хранят ссылки на файлы предыдущих версий программы. И, хотя сами файлы недоступны, можно выяснить их точные имена и попытаться выполнить поиск другими средствами.

Раньше The Wayback Machine позволяла производить поиск только по адресам страниц, но не по их содержимому. Зато для каждого адреса показывалось, когда была сохранена копия страницы и отличалась ли она от предыдущей версии. Но сейчас в рамках Wayback Machine появился сервис Recall (находящийся пока в стадии бета-тестирования), позволяющий выполнять полноценный текстовый поиск по 11 миллиардам сохраненных страниц.

The Wayback Machine можно найти по адресу **web.archive.org.**

20.2.5. FTP Search

Если известно точное имя файла, то его можно попытаться найти на FTP-сайтах. Но обходить сайты один за другим — занятие почти безнадежное. Гораздо эффективнее воспользоваться службой FTP Search, робот которой с некоторой периодичностью обходит все известные ему FTP-серверы и запоминает информацию о найденных файлах.

Таких служб довольно много. Программа ReGet Deluxe, например, умеет выполнять поиск файлов в семи каталогах FTP-сайтов:

- SunSITE (**sunsite.cnlab-switch.ch:8000**);
- FileSearch.ru (**www.filesearch.ru**);
- Lap Link (**ftpsearch.laplink.com**);
- Rambler (**ftpsearch.rambler.ru**);
- SUNET (**ftp.sunet.se:8000/ftpsearch);**
- FtpFind **(www.ftpfind.com);**
- FileMirrors **(www.filemirrors.com).**

20.2.6. Peer-to-Peer networks

Еще одно место, где можно найти почти все что угодно, — это пиринговые (Peer-to-Peer) сети. Первым широко известным их представителем был Napster, предназначенный для обмена музыкальными композициями. Сейчас существует довольно много разных файлообменных сетей, например eDonkey2000, iMesh или Kazaa.

В пиринговых сетях нет одного конкретного места, где хранится файл. Каждый пользователь делает общедоступной фрагмент своей файловой системы и передает информацию о находящихся там файлах на сервер. Другой пользователь может обратиться к серверу и узнать, у кого есть интересующий его файл. После этого устанавливается прямое соединение между двумя пользователями (без участия сервера), и выполняется передача данных. При этом разные фрагменты одного и того же файла могут скачиваться сразу у нескольких пользователей. И если один из пользователей, у которых этот файл был в наличии, отключается от сети, фрагмент может быть скачан у кого-то другого.

Разумеется, в пиринговых сетях можно найти только то, что кто-то из пользователей выложил в открытый доступ. Но старые версии программ там можно встретить довольно часто.

20.2.7. Распродажи

Еще один способ найти старые версии программ — купить их на распродаже. Иногда распродажи устраивают сами производители программ и отдают устаревшие версии по символическим ценам, например S 10 за каждую копию продукта, которая стоила в свое время $ 200.

Другой вариант найти уцененное программное обеспечение — воспользоваться онлайновым аукционом, например eBay. На таком аукционе легальный пользователь, решивший по каким-то причинам отказаться от использования программы, продает свою лицензию кому-то другому. И, разумеется, при этом он просит меньше денег, чем сам заплатил при покупке.

20.3. Саморазвитие и интеллектуальные игры

Кроме информации о конкретном продукте, в Интернете можно найти и знания. В своей работе исследователь защиты нередко сталкивается с ранее не встречавшимся и непонятными ему методами и приемами. И для того чтобы разобраться в особенностях их реализации, приходится многому учиться.

К счастью, в сети можно найти несколько сайтов, авторы которых собрали коллекцию задач разной сложности и организовали конкурс-игру по их решению. Иногда задания просто выдаются одно за другим, иногда цепь заданий даже связана каким-то сюжетом.

Задания могут относиться к самым разным областям: программированию на популярных языках (Assembler, C, Haskel, JavaScript, Perl, Python), сетевым

технологиям, математике, логике, криптографии и стеганографии, умению выполнять поиск в Интернете, работать с отладчиками и дизассемблерами и т. д.

Первые задания обычно очень простые и решаются неподготовленным человеком за считанные минуты. Решение заданий первого уровня позволяет получить доступ к следующему уровню и его заданиям. Каждый следующий уровень становится все труднее и требует все больше знаний, подталкивая тем самым участников игры к самообразованию.

Часто игре сопутствует форум, в котором разрешено обсуждать вопросы, касающиеся прохождения того или иного уровня, но нельзя просить и давать прямых подсказок по решению задач. И почти всегда можно узнать, на каком уровне находится каждый из участников, и понять, что почти всегда есть к чему стремиться.

Материальных призов в подобных играх почти никогда не бывает. Но участников, кроме рейтинга в общей таблице конкурсантов, ждет более ценная награда — знания и навыки, полученные в ходе решения конкурсных задач.

Разумеется, подобные игры интересны далеко не всем. Но, похоже, люди со склонностью к исследованиям получают огромное удовольствие от решения таких головоломок.

Тем, кто хочет попробовать свои силы, можно порекомендовать следующие проекты:

- Electrica the Puzzle Challenge (**www.caesum.com/game/index.php**);
- Resistor Challenge (**resistor.topgamers.net**);
- Mod-X (**www.mod-x.co.uk**);
- The Reverse-Engeneering-Academy (**www.reverser-course.de**).

Иногда и крупные компании организуют конкурсы, причем с настоящими призами. Много таких конкурсов устраивала RSA Data Security. Приз обычно составлял $ 10 000, а целью конкурса было подобрать ключ шифрования к данным, защищенным при помощи одного из криптографических алгоритмов, разработанных в RSA Data Security.

Производители оборудования и программного обеспечения иногда тоже объявляют конкурсы, в которых требуется получить контроль над общедоступным сервером, на котором установлен определенный набор программ. Как правило, набор программ совпадает с тем, что используется в реальных системах. И если за достаточно длительный период никто не сможет нанести работе сервера ощутимый урон, значит, подобную конфигурацию можно смело использовать на практике.

А компания Thawte в ноябре 2003 года объявила о начале четвертого Crypto Challenge, в рамках которого всем желающим предлагается взломать шифр,

разработанный специально для этого конкурса. В качестве награды первый человек, приславший правильное решение, должен получить цифровую фотокамеру Nikon, а десять следующих за ним — книгу Кевина Митника (Kevin Mitnick) "Искусство обмана" ("Art of Deception").

В целом, несмотря на плохую структурированность Интернета, при желании там можно почерпнуть практически все необходимые для проведения исследований знания, получить дополнительную информацию о предмете анализа и найти много полезных инструментов.

Глава 21



Инструментарий исследователя

При исследовании программ специалисту приходится пользоваться различными инструментами, позволяющими более эффективно выполнять поставленные задачи. Эти инструменты способны поднять производительность труда аналитика в несколько раз. О существующих инструментах и их возможностях полезно знать и разработчикам средств безопасности, чтобы более эффективно создавать трудности аналитикам, которые будут пытаться найти дыры в защите.

21.1. Классификация инструментов

Инструменты можно классифицировать по-разному. Например, на пассивные и активные.

Пассивные инструменты не оказывают никакого воздействия ни на саму исследуемую программу, ни на ее окружение. Активные инструменты, наоборот, взаимодействуют с программой во время ее выполнения. Из этого следует два важных замечания:

- активный инструментарий может дать гораздо больше информации, чем пассивный, т. к. позволяет оценивать состояние программы в динамике;
- присутствие активных инструментов может быть обнаружено защитой и может привести к ответным действиям. Обнаружить или предотвратить применение пассивных средств программа не в состоянии.

Активные инструменты, в свою очередь, могут использоваться только для протоколирования (мониторинга) хода выполнения программы или для явного воздействия на ход выполнения программы: подмены данных, исправления результатов проверки условий и т. д.

Также активный инструментарий может производить виртуализацию среды выполнения программы. То есть программа находится в полной уверенности, что выполняется в самых обычных условиях, а на самом деле каждый ее шаг, включая действия по поиску активных средств анализа, находится под полным контролем исследователя.

Еще один способ классификации инструментов — по области применения, т. е. что именно подвергается исследованию:

- исполняемый код;
- ресурсы приложения;
- дисковые файлы;
- записи в реестре;
- информация в оперативной памяти;
- информация, получаемая от устройств ввода;
- информация, посылаемая на устройства ввода;
- сообщения и данные, пересылаемые между процессами и внутри процесса;
- данные, передаваемые по сети;
- вызовы библиотечных функций.

Рассмотрим несколько наиболее мощных инструментов исследования.

21.2. Анализ кода программ

Два основных способа исследования программного кода — это дизассемблирование и отладка.

Используя дизассемблер, можно посмотреть, как устроена программа, какие команды и в какой последовательности должны выполняться, к каким функциям идет обращение и т. д. В общем случае дизассемблер не способен восстановить исходный текст программы, написанной на языке высокого уровня, таком как С или Pascal. Результатом работы дизассемблера является (как можно догадаться из названия) эквивалентный текст на языке ассемблера. Для осмысления ассемблерного текста аналитик, разумеется, должен быть хорошо знаком с языком ассемблера и с особенностями той среды, в которой должна выполняться дизассемблируемая программа. Дизассемблер является пассивным инструментом — он никак не воздействует на программу. Самым мощным дизассемблером из существующих на сегодняшний день можно смело назвать дизассемблер IDA Pro (Interactive DisAssembler), разработанный компанией DataRescue.

Для зашиты от дизассемблеров применяются различные методы. Например, если код программы запакован или зашифрован, дизассемблер не сможет увидеть в исследуемом файле настоящие инструкции и окажется бесполезен. Но защищенную таким образом программу можно сначала расшифровать и распаковать, а потом воспользоваться дизассемблером.

Для большинства популярных средств упаковки и шифрования кода исполняемых модулей давно разработаны автоматические или полуавтоматические

распаковщики. А для того чтобы узнать, чем именно запакован тот или иной модуль, можно воспользоваться специальными программами-идентификаторами, которые по некоторым характерным признакам способны опознавать название и версию используемого средства защиты, а также версию компилятора, применявшегося при разработке программы.

Так что реальную сложность для дизассемблирования представляют только программы, которые расшифровывают фрагменты кода динамически, не допуская единовременного присутствия в памяти расшифрованного кода целиком.

В некоторых случаях дизассемблер отказывается работать с исполняемым файлом, если какие-то заголовки файла сформированы с нарушением спецификации, но данный способ также не является надежным.

Иногда код программы модифицируется таким образом, чтобы дизассемблированную последовательность команд было очень трудно анализировать. Например, соседние команды разносятся в разные места, а правильность выполнения организуется за счет большого числа безусловных переходов. Или между командами вставляются произвольные фрагменты кода, не влияющие на результаты вычислений, но отнимающие у человека, выполняющего анализ, уйму времени.

Правда, стоит отметить, что, например, дизассемблер IDA Pro имеет весьма мощные средства расширения (подключаемые модули и язык сценариев), предоставляя тем самым возможность нейтрализовать все попытки противодействия дизассемблированию и последующему анализу.

Отладчик, в отличие от дизассемблера, является активным инструментом и позволяет проследить процесс выполнения по шагам, получая в любой момент всю информацию о текущем состоянии программы или вносить изменения в порядок ее выполнения. Разумеется, отладчик способен показывать дизассемблированные инструкции, состояния регистров, памяти и многое другое. Но наличие отладчика, в силу его активности, может быть обнаружено программой или той ее частью, которая отвечает за защиту. И программа может предпринять ответные действия.

Отладчики бывают трех основных типов: уровня пользователя, уровня ядра и эмулирующие.

Отладчики пользовательского уровня (User-level Debuggers) имеют практически те же возможности, что и отлаживаемая программа. Они используют Debugging API, входящий в состав операционной системы и с его помощью осуществляют контроль над объектом отладки. Отладчики пользовательского уровня входят в состав многих сред разработки, таких как Visual Studio. Они пригодны для исследования незащищенных программ, но могут быть легко обнаружены.

Отладчики уровня ядра (Kernel-mode Debuggers) встраиваются внутрь операционной системы и имеют гораздо больше возможностей, чем отладчики пользовательского уровня. Из ядра операционной системы можно контролировать многие процессы, не доступные другими способами. Одним из самых мощных и часто используемых отладчиков уровня ядра является SoftIce, разработанный в компании NuMega Labs (Compuware Corporation). Но и отладчики уровня ядра почти всегда могут быть обнаружены из программы, не имеющей доступа к ядру. Хотя для SoftIce, например, был разработан модуль расширения IceExt, позволяющий, среди прочего, неплохо скрывать наличие отладчика в памяти.

Эмулирующие отладчики, пожалуй, являются самым мощным средством исследования кода программ. Такие отладчики эмулируют выполнение всех потенциально опасных действий, которые программа может использовать для выхода из-под контроля исследователя. Однако основная проблема создания эмулирующих отладчиков заключается в том, что иногда им приходится эмулировать реальное периферийное оборудование, а это чрезвычайно сложная задача. Возможно поэтому сейчас нет доступных широкой аудитории эмулирующих отладчиков, хотя существует как минимум два пакета для создания виртуальных компьютеров: VMware, разработанный одноименной компанией, и VirtualPC, созданный в Connectix Corp. и недавно перешедший в собственность корпорации Microsoft.

Для защиты от отладки программа должна уметь определять наличие отладчика. Для обнаружения того же SoftIce разработано более десяти способов. Но в некоторых случаях можно определить, что программа исследуется при помощи отладчика по косвенным признакам, таким как время выполнения.

В современных процессорах с архитектурой x86 реализована команда RDTSC (Read Time-Stamp Counter). Эта команда позволяет получить количество тактов процессора, прошедших с момента включения питания или последнего сброса. Очевидно, что отладчик тоже является программой. Следовательно, когда защищенная программа исследуется отладчиком, изрядная часть тактов процессора расходуется на выполнение его кода. И если программа знает приблизительное количество тактов, необходимое для выполнения определенного фрагмента кода, то, измерив реально затраченное число тактов, легко обнаружить значительное увеличение времени выполнения, затраченного на отладку.

Для программ, компилируемых в псевдокод, также существуют и отладчики, и декомпиляторы, выдающие исходный текст не на ассемблере, а в некотором ином представлении, пригодном для анализа.

21.3. Работа с ресурсами

Для того чтобы найти какую-нибудь информацию в ресурсах исполняемого модуля, можно воспользоваться даже таким инструментом, как редактором Microsoft Visual Studio. Но лучше использовать специализированные редакторы ресурсов, которых существует довольно много. Эти редакторы, как правило, позволяют просматривать ресурсы известных типов (например текстовые строки, иконки, картинки и описания диалогов) в естественном виде, а незнакомые ресурсы — в виде шестнадцатеричного дампа.

Полезным может оказаться и модуль расширения Resource Browser, подключаемый к файловому менеджеру FAR. Этот модуль позволяет просматривать ресурсы в виде иерархического фрагмента файловой системы с подкаталогами и файлами. При использовании такого представления ресурсов очень удобно производить поиск.

21.4. Доступ к файлам и реестру

О программах-мониторах, протоколирующих попытки доступа к реестру и дисковым файлам, рассказывалось в *разд. 14.3.2.* Монитор реестра (Registry Monitor) и монитор доступа к файлам (File Monitor) — это активные инструменты.

А пассивные инструменты просто запоминают состояния реестра или файлов и по расхождениям позволяют определить, что именно изменилось.

Простейший способ обнаружить измененные файлы, не сохраняя их целиком, — подсчитать и запомнить значения хэш-функции от содержимого каждого файла до и после выполнения процесса, вносящего изменения, а потом сравнить два набора хэшей между собой. Именно на таком принципе строилась работа антивирусного монитора AdInf, функционировавшего под DOS.

Если удалось установить, какие именно файлы подвергаются изменению, их можно целиком заархивировать, а потом, после внесения изменений, сравнить старое и новое содержимое. Для этого можно воспользоваться специальными инструментами или утилитой FC (File Compare), входящей в состав Windows. FC позволяет сравнивать как двоичные, так и текстовые файлы.

С реестром работать не так удобно, как с файлами, из-за того, что реестр Windows представляет собой довольно сложную древовидную структуру. Зато объем данных, хранимых в реестре, сравнительно невелик — от силы несколько десятков мегабайт. Поэтому можно просто обойти все ветви реестра и сохранить значения в собственном формате. Одна из программ, позволяющих это сделать, — Advanced Registry Tracer (ART), разработанная компанией ElcomSoft Co. Ltd.

ART предоставляет пользователю возможность сохранить несколько "снимков" текущего состояния реестра. Затем отдельные снимки реестра можно попарно сравнивать, моментально получая списки добавленных, измененных и удаленных ключей и значений.

21.5. Содержимое оперативной памяти

Для доступа к памяти процесса можно использовать функции стандартного Win32 API. В операционных системах семейства NT некоторые процессы могут быть запущены с атрибутами безопасности, не позволяющими простым пользователям получать доступ к внутренностям процесса. Но это делается для того, чтобы защитить ядро операционной системы в многопользовательской среде. А в случаях исследования программ, как правило, пользователь может поставить себе любые права доступа и не встретит препятствий для доступа к памяти исследуемого процесса.

Существуют также специальные программы, позволяющие не просто сохранить фрагмент памяти на диск, но записать его в формате Portable Executable (PE). Такая операция называется получением дампа исполняемого файла и применяется для получения расшифрованной и распакованной версии исследуемой программы.

21.6. Устройства ввода и вывода

Устройства ввода-вывода невозможно исследовать пассивными средствами, зато обращения к ним можно протоколировать.

Программы для протоколирования нажатий на клавиатуру обычно называют клавиатурными шпионами и применяют для перехвата паролей, вводимых пользователем. Однако этот прием применяется или троянскими программами, или при попытке выведать секретную информацию у человека, но никак не при исследовании средств защиты.

Протоколирование ввода и вывода в COM- и LPT-порты может осуществляться, например, при помощи программы PortMon, разработанной Марком Русиновичем (Mark Russinovich) из компании SysInternals.

21.7. Сообщения Windows

Очень многие процессы внутри Windows управляются с помощью сообщений. Разумеется, существуют программы, позволяющие отслеживать и протоколировать, какие именно сообщения были переданы тому или иному процессу.

Одной из таких программ является Microsoft Spy++, входящая в состав Visual Studio. Spy++ позволяет из списка или интерактивно на экране выбрать окно, сообщения для которого необходимо отслеживать, и просмотреть его свойства. Можно также задать, какие именно сообщения должны протоколироваться и какие их атрибуты будут показываться. Протокол может сразу записываться в файл.

21.8. Сетевой обмен

Для перехвата данных, передаваемых по сети, используются специальные программы, называемые снифферами (sniffer). Как правило, снифферы способны перехватывать все сообщения, передаваемые между устройствами внутри физического сегмента сети, к которому подключен компьютер со сниффером.

Несмотря на то, что уже много лет существуют протоколы, позволяющие скрыть от противника всю важную информацию при передаче по сети, до сих пор не вышли из употребления некоторые протоколы, в которых, например, пароли пользователей передаются в открытом виде.

Так в оригинальной версии протокола FTP (File Transfer Protocol), описанной в RFC 765 (Request For Comment № 765) и датированной июнем 1980 года, и в обновленной версии протокола от октября 1985 года (RFC 959) описан только один метод аутентификации. При использовании этого метода пароль передается открытым текстом как аргумент команды PASS.

Аналогично, протокол POP3 (Post Office Protocol — Version 3), описание которого впервые было опубликовано в 1988 году (RFC 1081), при аутентификации передавал пароль пользователя только открытым текстом.

Позже и для **FTP,** и для POP3 были сделаны расширения протокола и добавлены несколько более безопасных методов аутентификации. Но до сих пор многие люди продолжают по разным причинам использовать аутентификацию, при которой пароли передаются открытым текстом. Например, настройки почтового клиента могли очень давно не обновляться — зачем, ведь все и так работает. А некоторые клиентские программы и серверы просто не поддерживают расширенные способы аутентификации. И почти всегда по умолчанию используется самый совместимый, а не самый безопасный метод подключения, а пользователям не приходит в голову его поменять — далеко не каждый знает, что при помощи сниффера получить пароль **FTP** или POP3, передаваемый открытым текстом, не составляет труда.

21.9. Вызовы библиотечных функций

Очень много информации о программе можно получить путем анализа обращений, которые она делает к библиотечным функциям. Например, при использовании протокола SSL (Secure Sockets Layer) применение сниффера не дает никаких результатов, т. к. все сообщения, передаваемые по сети, оказываются зашифрованы. Но под Windows большинство программ для доступа к сети, так или иначе, используют библиотеку wsock32.dll (Windows Socket 32-Bit DLL). И, перехватывая обращения к функциям из этой библиотеки, можно получить доступ к содержимому передаваемых и получаемых сообщений, не применяя сниффер.

Аналогичным образом можно перехватывать и протоколировать обращения к другим библиотекам, входящим в состав Windows или распространяемым вместе с исследуемой программой.

Существует несколько решений для разработчиков, позволяющих перехватывать обращения к библиотечным функциям, например библиотека Detours, созданная корпорацией Microsoft, и библиотека ApiHooks, разработанная человеком по имени Радим Пича (Radim "EliCZ" Picha). Также в Интернете можно найти и готовые программы, предназначенные для протоколирования обращений к библиотечным функциям, например APIS32 от Виталия Евсеенко и APISpy32 разработки Ярива Каплана (Yariv Kaplan).

Глава 22



Реконструкция криптографических протоколов

Так как практически все программные средства защиты в той или иной форме используют криптографические примитивы, полезно иметь эффективный способ анализа именно криптографической части приложений. Но прежде чем криптоаналитик сможет оценить, насколько стойким является реализованный в программе криптографический протокол, необходимо выяснить точную последовательность выполняемых протоколом действий. Далее описываются некоторые идеи, позволяющие восстановить последовательность применения криптографических примитивов в программе, т. е. реконструировать криптографический протокол.

22.1. Область применения

Поскольку хороших универсальных решений не бывает, ограничим класс программных продуктов, для которых будет проводиться реконструкция протокола. Исследуемая программа должна удовлетворять следующим требованиям:

- та часть программы, в которой реализован криптографический протокол, скомпилирована в машинный код, т. е. выполняться напрямую процессором, а не виртуальной машиной. Это позволит использовать эвристики, которые плохо работают (или не работают вообще), если применяются к псевдокоду. Но данное ограничение нельзя назвать очень строгим, т. к. криптографические примитивы, полностью реализованные на базе виртуальной машины, выполняются очень медленно и неприменимы для большинства практических задач;
- программа выполняется под одной из версий 32-битовой операционной системы Windows на процессоре x86. Это также не очень сильно сужает область возможного применения, ведь большинство персональных компьютеров сейчас имеют именно такую конфигурацию. К тому же почти все эвристики могут быть адаптированы к другим операционным системам и аппаратным платформам;

- исполняемый модуль, подвергаемый анализу, не запакован и не зашифрован. Если это не так, распаковку необходимо выполнить до того, как приступать к исследованиям;
- в программе используются опубликованные в открытых источниках криптографические алгоритмы. Данное требование будет удовлетворено с очень большой вероятностью, т. к. большинство разработчиков предпочитают использовать надежные алгоритмы, а надежность криптографического алгоритма подразумевает открытость его спецификации;
- программа разработана в рамках обычного процесса проектирования программного обеспечения, т. е. код оптимизирован с целью упрощения или повышения скорости выполнения, а не написан специально таким образом, чтобы усложнить его анализ;
- программа создана и распространяется с соблюдением всех законов, патентов и лицензий, под действие которых она попадает.

Несмотря на большое количество перечисленных требований, им удовлетворяет подавляющее большинство программ под Windows, использующих криптографию.

22.2. Идентификация криптографической библиотеки

Так как самостоятельная реализация криптографических примитивов — довольно сложная задача, можно предположить, что разработчики предпочли использовать одну из существующих криптографических библиотек. Если удастся определить, какая именно библиотека была использована при разработке программы, это даст довольно много информации.

В объектных файлах, поставляемых в составе библиотеки, как правило, присутствуют символические имена (названия функций и переменных), несущие смысловую нагрузку. И если исследователю удастся установить однозначное соответствие между фрагментами кода программы и кодом библиотеки, по именам можно будет догадаться, что делает та или иная часть программы.

Кроме того, библиотеки обычно поставляются вместе с подробной документацией, используя которую, исследователь сможет точно узнать, что делает та или иная функция.

А если библиотека распространяется в исходных текстах, то сразу же становятся доступны все детали реализации того или иного алгоритма.

Так как же узнать, что за библиотека использовалась? В этом могут помочь несколько идей.

Во-первых, лицензия на использование библиотеки может требовать, чтобы называние библиотеки упоминалось в самой программе или в сопроводительной документации, и тогда определить библиотеку не составит труда.

Во-вторых, лицензия может ограничивать область применения библиотеки, например только для некоммерческих приложений или только на территории США. Пользуясь этим, можно исключить из рассмотрения все библиотеки, которые не должны были применяться для создания конкретной программы из-за лицензионных ограничений. Хотя бывают и исключительные случаи, когда разработчики кому-то предоставляют специальную лицензию, отличающуюся от опубликованной в открытых источниках.

В-третьих, разные библиотеки имеют разный набор доступных алгоритмов. Таким образом, если в документации на программу сказано, что для защиты используется такой-то алгоритм и этот алгоритм реализован только в библиотеках одного разработчика (как было долгое время с RSA из-за патентных ограничений), останется совсем немного вариантов.

И, в-четвертых, почти в любой библиотеке есть присущие только ей текстовые или двоичные строки, по которым эта библиотека может быть идентифицирована. Так, например, при использовании библиотеки BSAFE в теле программы может присутствовать строка "bsafe" или "bcert". Библиотека SSLeay содержит строку "part of SSLeay", а библиотека RSAEURO — "Copyright (c) J.S.A.Kapp".

22.3. Идентификация криптографических примитивов

Если удалось установить, какая криптографическая библиотека была задействована при разработке программы, или если это не удалось, следующим этапом все равно будет идентификация криптографических примитивов. В том случае, если имеется доступ к библиотеке, процесс идентификации становится проще.

Разумеется, идентификация алгоритмов в полностью автоматическом режиме вряд ли возможна. И часто требуется участие аналитика, который знаком с различными алгоритмами и может по фрагменту дизассемблированного кода определить, какому примитиву он соответствует.

22.3.1. Идентификация функций по шаблонам

Наличие доступа к библиотеке, применявшейся в программе, позволяет выполнить автоматический поиск функций по шаблонам. Этот процесс состоит из двух этапов.

На первом этапе для каждой библиотечной функции, имеющей имя, создается шаблон. Для этого анализируются первые несколько байт функции и запоминаются значения тех байт, которые не будут изменяться редактором связей во время сборки программы. Изменяемые байты (как правило, это ссылки на данные и другие функции) в скомпилированной программе могут принимать любое значение, и в шаблоне они помечаются специальным образом.

После того как созданы шаблоны для всех функций, можно приступать непосредственно к идентификации. Для этого необходимо попытаться "приложить" шаблон каждой библиотечной функции к началу каждой функции в исследуемой программе. При совпадении всех неизменяемых байт шаблона функция считается опознанной. Однако необходимо учитывать, что несколько библиотечных функций могут иметь одинаковые шаблоны, как и один шаблон может соответствовать нескольким функциям в программе.

В дизассемблере IDA и сопутствующем ему инструментарии разработчика (Software Development Kit, SDK) реализованы средства, значительно облегчающие идентификацию функций. Для построения и удобного хранения шаблонов библиотек используется набор утилит FLAIR (Fast Library Acquisition for Identification and Recognition, быстрая обработка библиотек для идентификации и распознавания). Для распознавания функций применяется технология быстрой идентификации FLIRT (Fast Library Identification and Recognition Technology).

В FLAIR и FLIRT применено несколько интересных решений, позволяющих компактно хранить шаблоны и очень быстро оценивать соответствие им функций. При этом процент нераспознанных функций и, что более важно, процент неверно распознанных функций получаются очень низкими.

Предположительно, FLAIR и FLIRT основаны на работах Кристины Цифуентес (Cristina Cifuentes) и Майкла Ван Еммерика (Michael Van Emmerik).

22.3.2. Константы в алгоритмах

Если не удалось установить, какая библиотека использовалась при компиляции исследуемой программы, или нет возможности получить доступ к этой библиотеке, можно попытаться идентифицировать криптографические функции другим способом — по используемым константам.

Так, например, при инициализации многих хэш-функций (таких как MD4, MD5, SHA-1, RIPEMD-160) используются константы 0x67452301, 0xEFCDAB89, 0x98BADCFE и 0x10325476. В SHA-1 и RIPEMD-I60 используется также значение 0xC3D2ElF0.

В функции трансформации, используемой при вычислении SHA-1, применяются константы 0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC и 0xCA62C1D6. Эти константы являются представлением целой части чисел $2^{30}\times$Sqrt(2), $2^{30}\times$Sqrt(3), $2^{30}\times$Sqrt(5) и $2^{30}\times$Sqrt(10), где Sqrt(*x*) — функция извлечения квадратного корня из *x*.

В функции трансформации RIPEMD-160 последняя константа вместо 0xCA62C1D6 имеет значение 0xA953FD4E, что соответствует $2^{30}\times$Sqrt(7).

Функция трансформации MD5 использует 64 константы, вычисляемые как $2^{32}\times$Abs(Sin(*i*)), где *i* — номер раунда, от 1 до 64. Sin(*x*) вычисляет синус аргумента, заданного в радианах, а Abs(*x*) возвращает абсолютное значение *x* (без знака). Так, например, константы для первых четырех раундов равны 0xD76AA478, 0xE8C7B756, 0x242070DB и 0xC1BDCEEE соответственно.

При вычислении хэш-функции MD2 используется таблица перестановок (S-Box) размером 256 байт, начинающаяся с последовательности 0x29, 0x2E, 0x43, 0xC9, 0xA2, 0xD8, 0x7C, 0x01.

При шифровании по алгоритму RC5 используются две константы *P* и *Q*, значения которых основаны на двоичном представлении чисел *e* и π. Для версии RC5, работающей с 64-битовыми словами, эти константы имеют значения 0xB7E151628AED2A6B и 0x9E3779B97F4A7C15.

В спецификации некоторых алгоритмов, например RC4, нет вообще ни одной константы, позволяющей выполнять поиск (числа 256 и 0xFF, используемые при загрузке ключа и при шифровании, применяются настолько часто, что будут найдены и в сотнях посторонних функций). Однако если в программе используется оптимизированная версия RC4, подходящая константа может быть найдена. Дело в том, что процедура загрузки ключа начинается с того, что 256-байтовый массив заполняется последовательно числами от 0 до 255. Существует весьма эффективный способ выполнения данного цикла:

```
    lea   edi,data
    mov   eax,03020100h
    mov   edx,04040404h
    mov   ecx,64
setNext:
    stosd
    add   eax,edx
    loop  setNext
```

Как видно, использование оптимизации привело к появлению сразу двух констант, позволяющих выполнить идентификацию: 0x03020100 и 0x04040404.

Когда известно, какие константы присутствуют в том или ином алгоритме, остается найти эти константы в теле исследуемой программы. Поиск можно выполнять вручную или воспользоваться готовым инструментом, таким как CC (Crypto Checker), созданный человеком с псевдонимом Aleph, или KANAL (Krypto ANALyzer), разработанный группой uNPACKiNG gODS.

Crypto Checker 1.1 beta 7 умеет распознавать алгоритмы Blowfish, CAST-128, CAST-256, HAVAL, MARS, MD4, MD5, RC5, RC6, Rijndael, RIPEMD-128, RIPEMD-160, SHA-1, SHA-256, Tiger, Twofish, WAKE, а также некоторые генераторы псевдослучайных чисел, функции вычисления CRC16 и CRC32 и более 3000 простых чисел.

22.3.3. Принцип локальности

Если удалось найти хотя бы одну из использованных криптографических функций, обычно не очень сложно найти и все остальные. В этом очень помогает несколько эвристик.

Согласно первой эвристике все функции, относящиеся к одной библиотеке, редактор связей обычно располагает рядом. То есть, опознав один из криптографических примитивов, имеет смысл внимательно изучить функции, расположенные в непосредственной близости от него. С большой вероятностью это будут фрагменты других криптографических примитивов.

Вторая эвристика использует тот факт, что очень часто отдельные стадии алгоритма выполняются непосредственно друг за другом. То есть, например, при вычислении значения хэша используются три функции. Первая функция (`Init`) устанавливает начальное значение контекста. Вторая функция (update) обрабатывает очередную порцию данных, от которых вычисляется значение хэша и обновляется состояние контекста. Третья функция (Final) завершает процедуру вычисления и возвращает итоговое значение. И в реальной программе вызов функции `Init` обычно находится в непосредственной близости от первого вызова функции `Update`, а последний вызов update — прямо перед вызовом Final. Следовательно, найдя любую из этих функций, очень просто найти все остальные. На рис. 22.1 приведен пример процедуры, вычисляющей хэш-значение по алгоритму MD5. Функции `MD5_Init` и `MD5_Update` легко опознать по константам, а `MD5_Final` можно найти исходя из того, что она вызывается сразу после `MD5_Update`.

Третья эвристика очень помогает, если криптографический примитив реализован как класс в объектно-ориентированном языке. При компиляции класса создается таблица виртуальных функций (Virtual Function Table, VTable), содержащая адреса всех функций, являющихся методами данного класса. Следовательно, определив расположение одного из методов, можно найти ссылку на него из таблицы виртуальных функций, а значит, отыскать

и все остальные методы класса. На рис. 22.2 проиллюстрированы структуры данных класса, предназначенного для вычисления значения хэш-функции. Конкретный экземпляр класса вычисляет хэш-функцию MD5.

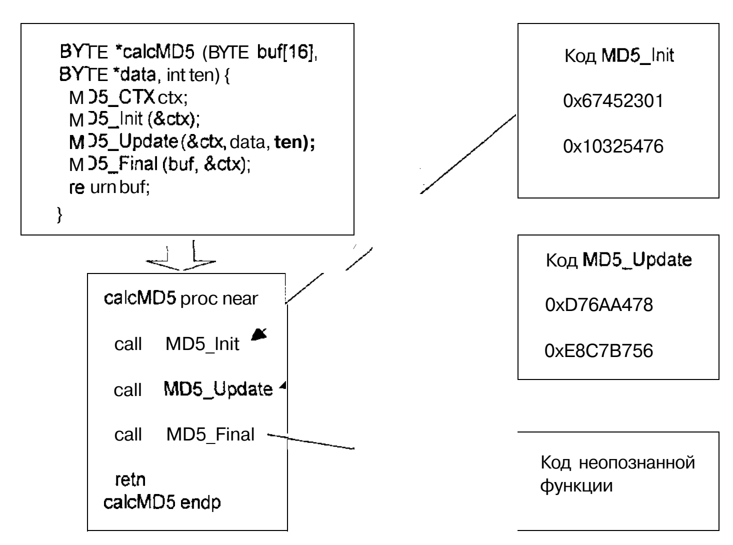


Рис. 22.1. Последовательное выполнение стадий алгоритма

Указатель на таблицу методов	Virtual Function Table	MD5::Init
	&Init	
Область данных экземпляра класса	«Update	MD5::Update
	&Final	
	NULL	MD5::Final

Рис. 22.2. Экземпляр класса, вычисляющего значение хэш-функции

Также если в программе поддерживаются, например, несколько симметричных алгоритмов шифрования, то с большой вероятностью где-то будет расположена таблица, каждая запись которой ссылается на таблицу виртуальных функций одного из алгоритмов.

22.4. Протоколирование

Когда определены адреса точек входа в каждую функцию, относящуюся к реализации того или иного криптографического примитива, необходимо

проанализировать все обращения к таким функциям. Разумеется, можно попытаться выполнить эту работу при помощи отладчика, но, скорее всего, число обращений к криптографическим функциям будет настолько велико, что удержать полную картину в памяти не сможет ни один человек.

Поэтому лучше сохранить всю информацию об аргументах, передаваемых на вход криптографических функций, а также о возвращаемых ими значениях, в файле протокола. Если исследуемая программа содержит несколько потоков, разумно запоминать и идентификатор потока, внутри которого происходит обращение к функции. Также можно сохранять адрес, откуда производился вызов той или иной функции.

Чтобы получить возможность протоколирования, необходимо перехватить все обращения к криптографическим функциям. Это может быть выполнено разными способами, например при помощи запуска программы под своим отладчиком, реализованным средствами Microsoft Debugging API.

Другой способ — модифицировать образ программы в памяти таким образом, чтобы при обращении к криптографическим примитивам управление поступало к специальным функциям-переходникам. Функция-переходник должна записать в протокол все аргументы запроса, вызвать оригинальную криптографическую функцию, к которой производится обращение, и сохранить в протокол возвращенные результаты. Обычно при реализации этого способа весь код, отвечающий за протоколирование, компилируется в виде отдельной динамически загружаемой библиотеки. А эта библиотека подключается к исследуемой программе с помощью технологии DLL Injection — внедрение DLL в адресное пространство процесса.

После того как протокол получен, остается его проанализировать. Протокол может быть представлен в виде ориентированного графа, в котором криптографические функции являются узлами, а принимаемые и возвращаемые значения — дугами.

Очень часто протокол строится таким образом, что данные с выхода одной криптографической функции сразу же подаются на вход другой, и так происходит до тех пор, пока не будет получен некоторый результат. То есть модификация данных происходит только внутри криптографических функций.

В таком случае, зная конечный результат (который, например, мог быть найден в файле на диске или перехвачен с помощью сниффера), получить алгоритм его вычисления не сложно. Достаточно найти на графе, построенном по информации из протокола, обратный путь до исходных данных.

22.5. Внесение искажений

Если же данные все-таки изменяются между вызовами криптографических функций и это не позволяет определить алгоритм, можно попробовать следующий подход. Некоторые функции-переходники разрабатываются таким образом, что при обнаружении определенного значения в обрабатываемых данных, в них вносятся намеренные искажения. При этом, разумеется, все операции, использующие искаженные данные, пойдут по-другому, и это будет отражено в протоколе. Сравнив два протокола (оригинальный и с искажениями), можно будет легко определить, где внесенные искажения впервые повлияли на ход выполнения протокола, а значит, и локализовать место, нуждающееся в более подробном исследовании с помощью отладчика и дизассемблера.

Изучать и анализировать реконструированный криптографический протокол на несколько порядков проще, чем пытаться при помощи отладчика и дизассемблера разобраться в том, как программа обрабатывает данные.

Если же протокол является единственной секретной частью "черного ящика", реконструкция протокола, фактически, означает полный взлом защиты.

Глава 23



Чего ожидать в будущем

Делать прогнозы на будущее — дело неблагодарное. Но то, что написано в этой главе, — это не столько прогнозы, сколько общие ощущения от существующего на настоящий момент положения вещей и намечающихся тенденций.

23.1.Концепциибезопасности

К счастью, все больше компаний приходят к пониманию того, что в современном информационном мире обеспечение информационной безопасности является очень важной задачей. Возможно, этому способствует тот факт, что все больше инцидентов, связанных с нарушением защиты, становятся известными широкой аудитории. И в этом довольно большая заслуга средств массовой информации, которые стали уделять внимание подобным вопросам.

Правда, иногда возникает ощущение, что средства массовой информации намеренно забывают некоторые факты, стремясь раздуть скандал. Иначе как объяснить, что очень много кричат о компьютерном вирусе, вызвавшем эпидемию, хотя заплатка для уязвимости, использованной вирусом, была доступна за несколько месяцев до появления вируса. То есть эпидемия случилась исключительно из-за халатности системных администраторов, не установивших вовремя обновление системы безопасности.

В любом случае, очень многие люди, соприкасающиеся с информационной безопасностью не только как пользователи, имеют недостаточный багаж знаний в этой области. И прежде чем вкладывать весьма значительные средства в разработку или приобретение средств защиты, стоит обучить людей хотя бы основным принципам безопасности, т. к. без этого даже самая надежная техническая система будет давать сбои в самом слабом звене — человеческом факторе.

23.2. Перспективы развития криптографии

Как бы ни было много сделано, ни одна из наук не собирается останавливаться в своем развитии. Так и в области криптологии постоянно ведутся исследования. Часть проводимых работ относится к криптоанализу — вопросами проверки стойкости алгоритмов и поиском методов их взлома занимаются ведущие мировые криптографы. Но не прекращаются и усилия по созданию новых методов для защиты информации.

23.2.1. Потребность в новых криптографических примитивах

Несмотря на то, что существующие криптографические алгоритмы способны обеспечить достаточно высокий уровень безопасности, чтобы защитить данные от любого противника на сотни лет, новые шифры продолжают появляться. Так сравнительно недавно появилась группа неплохих алгоритмов, ставших финалистами конкурса AES.

Иногда новые алгоритмы должны работать в специальных условиях (мало памяти, ограниченный набор команд), иногда требуется увеличить производительность без снижения стойкости. Работы по созданию новых симметричных шифров ведутся постоянно, но значительного изменения состава широко применяемых симметричных криптографических алгоритмов, наверное, уже не произойдет. Все-таки симметричные шифры — одна из самых древних и хорошо изученных областей криптографии.

А вот в криптографии с открытым ключом до сих пор много чего не сделано. Хорошо проверенные методы, такие как RSA, требуют выполнения значительных объемов вычислений и оперируют блоками большого размера. И с увеличением минимальной рекомендованной длины ключа вследствие прогресса вычислительной техники и методов взлома накладные расходы растут очень быстро. Так что поиск более технологичных решений, способных обеспечить высокий уровень безопасности, может, в конце концов, привести к появлению принципиально новых алгоритмов.

128- и 160-битовые значения, получаемые на выходе широко используемых хэш-функций MD5 и SHA-1, оказались слишком короткими для некоторых приложений, и были разработаны спецификации SHA-256, SHA-384 и SHA-512. Дальнейшее увеличение размера хэша вряд ли найдет практическое применение, но могут появиться хэш-функции, работающие быстрее, чем SHA.

Еще одна из плохо проработанных задач — это источники случайности для генераторов псевдослучайных чисел. Но поиск новых источников вряд ли

относится к задачам криптографии. А вот оценка объема действительно случайной информации, получаемой из каждого источника, вполне заслуживает исследования.

23.2.2. Надежные, но не всегда работающие протоколы

С протоколами дело обстоит гораздо хуже, чем с алгоритмами. Есть еще много областей, в которых протоколы существуют, но не всегда справляются с возложенными задачами.

Возьмем, например, цифровую подпись. Казалось бы, если существует инфраструктура открытых ключей и используются стойкие асимметричные алгоритмы, нет никакой проблемы в обеспечении проверки подлинности подписи, и подпись является неотказуемой.

Однако на самом деле возможен такой сценарий. На компьютере пользователя хранится секретный ключ, используемый для подписи, но для доступа к ключу необходимо ввести секретную фразу. Это очень распространенная схема хранения секретного ключа. Компьютер заражается вирусом, или на него попадает троянская программа. Эта троянская программа находит на машине ключ подписи и отсылает его по Интернету противнику. Параллельно устанавливается программа, протоколирующая все нажатия кнопок на клавиатуре и передающая протокол злоумышленнику. Таким образом, после того как ничего не подозревающий пользователь введет ключевую фразу, в распоряжении противника будет чужой ключ подписи и секретная фраза, необходимая для доступа к ключу.

Описываемый пример более чем реален, учитывая многочисленные дыры, с завидным постоянством обнаруживаемые в операционных системах. И как результат, цифровая подпись может быть подделана злоумышленником, и в этом нет прямой вины пользователя.

Теперь посмотрим на ту же ситуацию с другой стороны. Если владельцу ключа удастся доказать в суде, что похищение имело место, то это позволит ему освободиться от обязательств и утверждений, под которыми была поставлена его подпись. И достаточно одного подобного случая в стране, где действует прецедентное право (например в США), чтобы любой желающий получил хорошие шансы отказаться от подписи, изобразив похищение собственного ключа.

Троянская защита

17 октября 2003 года 19-летний Аарон Кафферей (Aaron Caffrey) был оправдан британским судом. Его обвиняли в организации DOS-атаки (Denial Of Service,

отказ в обслуживании) на инфраструктуру порта в Хьюстоне (штат Техас). Однако защитнику удалось убедить присяжных, что компьютер Аарона, с которого выполнялась атака, был взломан неизвестным хакером, который и устроил атаку против хьюстонского порта. И это как минимум третий случай успешного применения "троянской защиты" в британской юриспруденции.

Вся проблема в том, что для создания обычной подписи требуется присутствие подписывающего. А для цифровой подписи требуется доступ к некоторой информации или оборудованию (файлу с ключом, секретной фразе, смарт-карте, PIN-коду), находящимся в эксклюзивном распоряжении владельца подписи. Но все эти необходимые для подписи сущности могут быть отделены от пользователя: подсмотрены, украдены, взяты на короткое время и использованы без ведома владельца.

Чтобы "привязать" пользователя к проставляемой им подписи, можно использовать биометрику. Но и тут нерешенных проблем хватает.

Поддельные пальчики

Голландские специалисты по биометрике Тон ван дер Путте (Ton van der Putte) и Джерон Кенинг (Jeroen Keuning) к 2000 году разработали технологию, позволяющую обманывать сканеры отпечатков пальцев. Подделку не обнаруживал ни один из протестированных сканеров, созданных двумя десятками различных производителей.

В октябре 2003 года эксперимент был повторен, и результаты ошеломили даже авторов технологии. Комплект материалов, достаточный для изготовления примерно 20 поддельных пальцев, можно приобрести примерно за $ 10 в магазинах типа "сделай сам" (do-it-yourself). Изготовление копии пальца при сотрудничестве владельца занимает не более 15 минут. Более того, для изготовления подделки на основе латентного отпечатка (оставленного на гладкой поверхности) требовалось 1,5 часа времени, материалы на $20 (достаточные для 20 дубликатов) и такое "специальное" оборудование, как цифровая камера и ультрафиолетовая лампа.

Так что проблемы персональной аутентификации, а также многие другие проблемы современного информационного мира все еще нуждаются в более надежных решениях.

Кстати, многие неплохие алгоритмы и протоколы покрываются патентами или патентными заявками. И одно из занятий, на которое криптографам приходится тратить время из-за существующего патентного законодательства, заключается в поиске эффективных и надежных решений, не нарушающих никаких патентов.

23.3. Защита программ

Полностью защитить программу от несанкционированного тиражирования, применяя только программные решения, невозможно. Если программа может быть запущена, она может быть взломана.

Однако существуют идеи, способные значительно затруднить работу противника. В середине 2000 года в конференции новостей **fido7.ru.crypt** было опубликовано сообщение, автором которого являлся человек под псевдонимом stpark. В сообщении перечислялось несколько интересных методов, разработанных специалистами по защите и анализу программ для собственных нужд, не получивших открытой реализации и, возможно, именно поэтому не взломанных. Далее приведены три из них:

- перекрестная проверка целостности исполняемого модуля и используемых им динамически загружаемых библиотек;
- защита, выполняющаяся одновременно в нескольких потоках, где каждый поток контролирует целостность кода программы, выявляет непредусмотренные задержки в выполнении других потоков и постоянно изменяет внутреннее состояние модуля защиты;
- применение виртуальных машин для выполнения специальным образом обработанного кода.

Еще одна идея, которую собиралась реализовать (а может быть, уже и реализовала) компания Protection Technology, заключалась в разработке специального компилятора языка С, который бы создавал код, очень сложный для дизассемблирования.

Смысл этой идеи в том, что даже простейшие операции можно записать таким образом, что будет далеко не очевидно, что же они делают. И это очень часто получается при включении оптимизации в компиляторе. Например, эквивалентный ассемблерный текст следующей простейшей функции на языке С приведен в листинге 23.1:

```
int divFn (int x) { return x / 10; )
```

Данная функция выполняет одну-единственную операцию — целочисленное деление аргумента на 10.

Листинг 23.1. Функция целочисленного делении на 10

```
mov     ecx, x
mov     eax, 66666667h
imul    ecx
mov     eax, edx
```

```
sar     eax, 2
mov     ecx, eax
shr     ecx, 1Fh
add     eax, ecx
retn
```

А вот пример другой функции, ассемблерный код которой приведен в листинге 23.2:

```
int caseFn (int x) { return x > 100 ? 15 : 25; }
```

Эта функция в зависимости от значения аргумента возвращает одно из двух возможных значений.

Листинг 23.2. Функции выбора результата по значению аргумента

```
mov     ecx, [esp+arg_0]
xor     eax, eax
стпр    ecx, 64h
setle   al
dec     eax
and     al, 0F6h
add     eax, 19h
retn
```

Обе ассемблерные функции, приведенные выше, можно написать гораздо короче и понятнее, но при оптимизации по скорости выполнения именно эти варианты являются наилучшими. В первом примере удалось избавиться от очень медленной операции деления, а во втором — от команды условного перехода, также изрядно влияющей на скорость. Но оптимизацию выполнил компилятор, а человеку с первого взгляда совсем не просто будет понять, что делает каждая из функций. Хотя, немного подумав, разобраться все-таки реально. К тому же, существуют учебники, из которых можно узнать о хитростях, используемых при оптимизации, и научиться их понимать.

Но почти для любой конструкции языка подобным образом можно придумать несколько альтернативных способов представления в системе команд микропроцессора. И если компилятор станет случайным образом выбирать один из многих возможных вариантов для каждого оператора, разобраться в порождаемом им машинном коде человеку будет очень и очень непросто.

23.4. Защита данных

Несмотря на то, что на рынке полно откровенно плохих средств защиты (которые, тем не менее, часто хорошо продаются), с обеспечением секретности научились приемлемо справляться многие разработчики. И все чаще средства зашиты, встраиваемые в архиваторы и электронные таблицы, программы ведения финансовой истории и текстовые процессоры, позволяют предотвратить раскрытие секретной информации даже самым сильным противником. Но беда в том, что пользователи не приучены правильно использовать функции защиты. Для большинства простых людей предложение выбрать криптопровайдера, который будет использоваться для защиты данных, равносильно предложению выбрать марку стали, из которой будут делать гайки для закрепления двигателя автомобиля.

Здесь может быть два решения: или обучать пользователей, донося до них правильное понимание того, что такое безопасность, или построить защиту таким образом, чтобы пользователь просто не смог использовать короткий ключ шифрования или легко подбираемый пароль. Но, к сожалению, ни одно из этих решений работать не будет — не все пользователи хотят изучать то, что они сами считают ненужным. А применение жестких политик безопасности снижает совместимость и удобство использования (чем не все готовы пожертвовать), а также порождает другие виды уязвимостей, например пароли, записанные на бумаге и приклеенные к системному блоку.

С защитой цифрового контента ситуация более сложная. Задачи DRM являются сравнительно новыми, но практически все попытки их решения оказались неудачными. Частично это была вина разработчиков, частично причиной явилась невозможность абсолютной зашиты данных при DRM.

Владельцы контента пытаются искать самые разные подходы к обеспечению защиты. Потерпев неудачи в применении технических методов, они пустили в ход законодательные. В дополнение к законам, запрещающим тиражирование информации, защищенной авторскими правами, появились законы, по которым даже исследование технических средств защиты контента может быть признано уголовным преступлением. Медиамагнаты пытаются провести и новые законы, по которым каждое электронное устройство должно будет иметь встроенный блок, отвечающий за контроль соблюдения цифровых прав. А устройства, не имеющие подобного блока контроля (а это, например, почти все персональные компьютеры), должны быть признаны незаконными.

Можно понять желание продавцов контента любым способом собрать столько денег, сколько в состоянии заплатить пользователи, но для этого

совершенно необязательно превращать универсальный компьютер в узкоспециальное устройство для продажи звука и изображения. Можно попробовать и другие модели.

Так еще несколько лет назад озвучивались идеи продавать электронные книги не "в темную", как это делается сейчас (сначала заплати, а потом разбирайся, нужна тебе эта книга или нет), а авансом. Например, первую главу можно прочитать бесплатно. Но чтобы получить возможность читать следующую главу, необходимо оплатить предыдущую.

И не стоит делать таких жестких и порою бессмысленных ограничений на использование тех же электронных книг. Пока электронные книги не станут действительно удобными, мало кто будет их покупать. Уж лучше контролировать пути нелегального распространения с помощью стеганографии. А в этом направлении сделано пока не очень много.

23.5. Методы анализа программ

Основное преимущество исследователя перед разработчиком — неограниченность времени. В распоряжении разработчика есть период с момента запуска проекта и до момента выхода готовой программы. Этот период может длиться 2 недели, а может 3 года. Но именно за это время необходимо продумать и реализовать все необходимые средства защиты.

Исследователь получает систему защиты в свое распоряжение после того, как разработка завершена. И, начиная с этого момента, он может пробовать самые разные подходы для того, чтобы найти в защите слабое место. Исследователь не ограничен во времени — с программой ничего не случится ни через год, ни через 10 лет. В крайнем случае, всегда можно отключить компьютер от сети, отвести часы назад и восстановить конфигурацию системы с предварительно сохраненной резервной копии.

Разработчик может выпускать новые версии программ, добавляя или модифицируя их защитные механизмы, но он не может исправить уже существующую версию программы. А его противник, исследователь, имеет возможность обновлять свой инструментарий и навыки до тех пор, пока не найдет правильный подход.

То есть, несмотря на то, что разработчик всегда может оказаться впереди исследователя (о чем очень любят заявлять некоторые производители средств защиты от несанкционированного тиражирования программного обеспечения), их преимущество всегда носит локальный, временный характер. Для каждой защиты рано или поздно отыскивается метод противодействия, а программы, использующие такую защиту, оказываются взломаны.

И если разработчиком зачастую руководят только материальные стимулы, то исследователи почти всегда движимы любопытством, интересом. А для увлеченного человека это очень сильный мотив. Так что новые методы анализа будут появляться ничуть не медленнее, чем методы защиты.

С учетом всего вышесказанного, хочется пожелать удачи тем, кто пытается найти свое место в сфере информационной безопасности. По большому счету, стать специалистом по защите информации не так уж и сложно. Достаточно понимать, что можно делать с информацией, владеть современными технологиями безопасности, знать методы работы противника (независимо от того, на какой вы стороне) и никогда не переставать изучать, исследовать, докапываться до сути.

Благодарности

Книга закончена, и хочется сказать спасибо всем тем, кто способствовал ее скорейшему появлению на свет, а именно:

руководителям и работникам компании ElcomSoft Co. Ltd. **(www.elcomsoft.cora)** за моральную поддержку, практическую помощь и возможность писать книгу в рабочее время;

кафедре информационной безопасности МГТУ им. Баумана **(www.iu8.bmstu.ru)** за создание идеальных условий для преподавания;

работникам компании SmartLine, Inc. **(www.protect-me.com)** Ашоту Оганесяну (Ashot Oganesyan) за консультации в NT Drivers Development и Станиславу Винокурову (Stanislav Vinokurov) за подборку информации о протекторах;

посетителям форума **Reversing.net** за сложные вопросы и интересные ответы;

Брюсу Шнайеру (Bruce Schneier), **www.counterpane.com,** за замечательные книги, выпуски CRYPTOGRAM и тяжелую работу по популяризации идей криптографии и информационной безопасности;

Эрику Янгу (Eric Young) и Тиму Хадсону (Tim Hudson) за великолепную криптографическую библиотеку SSLeay, распространяемую в исходных текстах;

работникам издательства БХВ-Петербург, сделавшим все возможное для того, чтобы эта книга попала в руки читателей;

читателям, для которых писалась эта книга и которые, надеюсь, нашли в ней для себя много интересного и полезного.

Список использованных источников

1. Biham E., Kocher P. "A Known Plaintext Attack on the PKZIP Stream Cipher". Fast Software Encryption 2, Proceedings of the Leuven Workshop, LNCS 1008, December 1994.
2. Cerven P. Crackproof Your Software — The Best Ways to Protect Your Software Against Crackers — San Francisco: NO STARCH PRESS, 2002 — 272 pages.
3. Ferguson N., Schneier B. Practical Cryptography — John Wiley & Sons, 2003 - 432 pages.
4. Menezes A. J., van Oorschot P. C, Vanstone S. A. Handbook of Applied Cryptography — CRC Press, 1996 — 816 pages.
5. Вернет С, Пэйн С. Криптография. Официальное руководство RSA Security. — М.: Бином-Пресс, 2002 — 384 с.
6. Зегжда Д. П., Ивашко А. М. Основы безопасности информационных систем. — М.: Горячая линия — Телеком, 2000 — 452 с.
7. Иванов М. А. Криптографические методы защиты информации в компьютерных системах и сетях. — М.: КУДИЦ-ОБРАЗ, 2001 — 368 с.
8. Нечаев В. И. Элементы криптографии (Основы теории защиты информации): Учеб. пособие для ун-тов и педвузов / Под ред. В. А. Садовничего — М.: Высш. шк., 1999 — 109 с.
9. Чмора А. Л. Современная прикладная криптография. 2-е изд., стер. — М.: Гелиос АРВ, 2002 — 256 с.
10. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — М.: Издательство ТРИУМФ, 2002 — 816 с.
11. **http://aspack.com/asprotect.html** — ASPACK SOFTWARE— Best Choice Compression and Protection Tools for Software Developers.
12. **http://cryptome.org/ms-drm-os.htm** — CRYPTOME. Microsoft Digital Rights Management Operating System — US Patent No. 6,330,670.

13. **http://news.bbc.co.uk/2/hi/technology/3202116.stm** — BBC News. Questions cloud cyber crime cases.

14. **http://pilorama.com.ru/library/rflirt.html** — Ilfak Guilfanov. FLIRT — Fast Library Identification and Recognition Technology.

15. **http://research.microsoft.com/sn/detours** — Microsoft Research. Detours.

16. **http://resistor.topgamers.net** — khacrez. Resistor Challenge.

17. **http://retro.icequake.net/dob** — Ryan Underwood. The Central Point Option Board.

18. **http://securityhorizon.com/whitepapers/archives/lanman.html** — Brian. NT / LANMAN Password Security Discussion.

19. **http://triade.studentenweb.org/GInt/gint.html** — Triade systems — GInt Page.

20. **http://uozp.akcentplus.ru/16.htm** — Общества защиты прав потребителей г. Уфы. Статья 16. Недействительность условий договора, ущемляющих права потребителя.

21. **http://www.ahead.de/en/** — Ahead Software. Nero.

22. **http://www.anticracking.sk/elicz/export.htm** — EliCZ's Export (ApiHooks 5.6).

23. **http://www.arjsoftware.com** — ARJ Software, Inc.

24. **http://www.atstake.com/research/advisories/1999/95replay.txt** — weld@10pht.com. Win95/98 File Sharing Impersonation.

25. **http://www.average.org/freecrypto** — Использование "сильной" криптографии в России.

26. **http://www.cacr.math.uwaterloo.ca/hac** — Centre for Applied Cryptographic Research. Handbook of Applied Cryptography.

27. **http://www.caesum.com/game/index.php** — Cronos. Electrica the Puzzle Challenge.

28. **http://www.casi.org.uk/discuss/2003/msg00457.html** — Glen Rangwala. [casi] Intelligence? the British dossier on Iraq's security infrastructure.

29. **http://www.computer.org/security/garfinkel.pdf** — Simson L. Garfinkel, Abhi Shelat. Remembrance of Data Passed: A Study of Disk Sanitization Practices.

30. **http://www.computerbytesman.com/privacy/blair.htm** — Richard M. Smith. Microsoft Word bytes Tony Blair in the butt.

31. **http://www.convertlit.com** — Dan A. Jackson. Convert LIT.

32. **http://www.cryptonessie.org** — New European Schemes for Signature, Integrity, and Encryption.

33. **http://www.cs.berkeley.edu/~daw/my-posts/netscape-cracked-0** — Ian Goldberg, David Wagner. Netscape SSL implementation cracked!

34. **http://www.cyberlaw.com/rsa.html** — Patrick J. Flinn and James M. Jordan III. CyberLaw Presents: The RSA Algorithm & The RSA Patent.

35. **http://www.din.de/ni/sc27/** - ISO/IEC JTC 1/SC 27 - IT SECURITY TECHNIQUES.

36. **http://www.distributed.net/pressroom/news-20020926.txt** — David McNett. distributed.net completes rc5-64 project.

37. **http://www.ebookpro.com** — Internet Marketing Center. eBook Pro — Your Internet Publishing Solution.

38. **http://www.ebxwg.org** — EBX Workgroup (Open eBook Forum).

39. **http://www.eetimes.com/story/90193** - EE Times. WinZip Hits 100 Million Download Mark on CNET Download.com.

40. **http://www.elby.ch/en/products/clone_cd/** — Elaborate Bytes CloneCD.

41. **http://www.heise.de/tp/english/inhalt/te/2898/l.html** — Duncan Campbell. Export version of Lotus Notes provides trapdoor for NSA.

42. **http://www.heise.de/tp/english/inhalt/te/5263/1.html** — Duncan Campbell. How NSA access was built into Windows.

43. **http://www.honeynet.org/scans/scan24/sol/pedram/reference/mike_zipattacks.htm** — Michael Stay. ZIP Attacks with Reduced Known-Plaintext.

44. **http://www.intertrust.com/main/ip/litigation.html** — InterTrust Technologies — Litigation Status.

45. **http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html** — Information-technology Promotion Agency. Japan CRYPTREC.

46. **http://www.mail-archive.com/cryptography-digest@senator-bedfellow.mit.edu/msg04871.html** — cryptography-digest: Arcfour in Ada, by me — is it good?

47. **http://www.microsoft.com/technet/security/bulletin/MS01-017.asp** — Microsoft TechNet. Erroneous VeriSign-Issued Digital Certificates Pose Spoofing Hazard.

48. **http://www.mod-x.co.uk** — Mod-X Challenge.

49. **http://www.nist.gov/aes** — NIST. AES Home Page.

50. **http://www.nytimes.com/library/world/mideast/041600iran-cia-index.html** — James Risen. New York Times Special Report: The C.I.A. in Iran.

51. **http://www.paceap.com** — PACE Anti-Piracy.

52. **http://www.password-crackers.com/crack.html** — Pavel Semjanov. Russian Password Crackers.

53. **http://www.password-crackers.com/publications/crypto.html** — Павел Семьянов. Почему криптосистемы ненадежны?

54. **http://www.password-crackers.com/publications/crypto_eng.html** — Pavel Semjanov. On cryptosystems untrustworthiness.

55. **http://www.pcworld.com/news/article/0,aid,109720,00.asp** _ Mike Hogan. Intuit Sued Over Product Activation.

56. **http://www.planetpdf.com/mainpage.asp?webpageid=2434** — Kurt Foss. Washington Post's scanned-to-PDF Sniper Letter More Revealing Than Intended.

57. **http://www.planetpdf.com/mainpage.asp?webpageid=2450** — Planet PDF. U.S. Department of Justice selects Appligent Redax for PDF redaction.

58. **http://www.planetpdf.com/mainpage.asp?webpageid=3177** — Kurt Foss. Makeshift PDF Redaction Exposes 'Secret' Government Info — Again.

59. **http://www.planetpdf.com/mainpage.asp?webpageid=808** — Kurt Foss. PDF Secrets Revealed.

60. **http://www.rarlab.com** — RARLAB. WinRAR archiver, a powerful tool to process RAR and ZIP flies.

61. **http://www.reverser-course.de** — Zero, SantMat. The Reverse-Engeneering-Academy.

62. **http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html** — RSA Laboratories. Factorization of RSA-155.

63. **http://www.schneier.com/book-applied.html** — Schneier.com: Applied Cryptography by Bruce Schneier.

64. **http://www.sealedmedia.com** — SealedMedia — Complete document protection and control, even after delivery.

65. **http://www.siliconrealms.com/armadillo.shtml** — Silicon Realms. The Armadillo Software Protection System.

66. **http://www.slysoft.com/en/clonecd.html** — SlySoft — **CloneCD.**

67. **http://www.ssl.stu.neva.ru/psw/crypto/appl_rus/appl_cryp.htm** — Павел Семьянов. Брюс Шнайер. Прикладная криптография.

УДК 681.3.06
ББК 32.973.26-018.2
С43

Скляров Д. **В.**

С43 Искусство защиты и взлома информации. — СПб.: БХВ-Петербург, 2004. - 288 с: ил.

ISBN 5-94157-331-6

Защита информации — очень сложная наука, но начинать ее изучение можно с простых вещей. Именно так и задумана эта книга. Читателю предстоит узнать, чем занимается информационная безопасность и какие методы она использует для решения своих задач. Особое внимание уделяется криптографии — пожалуй, самому мощному инструменту защиты. Подробно рассматриваются вопросы защиты программ от несанкционированного тиражирования, а также различные аспекты обеспечения безопасности данных, включая управление цифровыми правами и применение стеганографии. Изложение материала сопровождается примерами неудачных средств зашиты и объяснением причин их появления. Рассказывается также об анализе средств зашиты, целях, которые ставятся при проведении анализа, и инструментах, применяемых при исследовании.

Для широкого круга пользователей, интересующихся вопросами защиты информации

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Григорий Добин*
Редактор	*Нина Седых*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Инна Тачина*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.12.03.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 23,2.
Тираж 3 000 экз. Заказ № 1310
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар Ns 77.99.02.953.Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-331-6